博士論文 2022 年度

実践的な認証認可基盤の実現のための管理運用統合プレーンアーキテクチャ

慶應義塾大学
大学院メディアデザイン研究科

加藤 大弥

本論文は慶應義塾大学大学院メディアデザイン研究科に
博士 (メディアデザイン学) 授与の要件として提出した博士論文である。

加藤 大弥

研究指導コミッティ：

砂原 秀樹 教授　(主指導教員)
南澤 孝太 教授　(副指導教員)
加藤 朗 教授　(副指導教員)

論文審査委員会：

加藤 朗 教授　(主査)
南澤 孝太 教授　(副査)
大川 恵子 教授　(副査)
鍛 忠司 博士　(副査、株式会社 日立製作所)

博士論文 2022 年度

実践的な認証認可基盤の実現のための管理運用統合プレーンアーキテクチャ

カテゴリ：サイエンス / エンジニアリング

論文要旨

インターネット上のサービスを利活用して生活している今日において、サービスへの Digital Identity の登録とそれに伴う認証は必要不可欠な要素である。しかし、Digital Identity の管理運用を行うための環境が整備されていないことを原因とする個人情報の漏洩や不正アクセス、認証認可機能の管理運用不備におけるサービス停止の問題が後を絶たない。本研究では、この課題を既存の手法の認証認可を取り巻くシステムの管理運用が困難であると捉え現状の解決を図る。

解決のために本研究では、認証認可基盤における管理運用統合プレーンアーキテクチャの定義を行う。プレーンとは特定の処理が行われる場所、機能を意味しており、通信機器などでは、データを転送するために使用される機能の処理を行う層をデータプレーン、データの送信と接続制御の機能の処理を行う層を制御プレーンとして定義することでネットワーク全体を効率よく管理運用している。本研究ではこの考え方を活用して、認証認可基盤における管理運用統合プレーンを定義することで運用者にとって実践的な認証認可基盤を実現することを研究の目的とする。

本研究では管理運用統合プレーンアーキテクチャを実現するために、まず現状の認証認可を伴うサービスのシステムアーキテクチャと問題が発生する要因を調査した。これによりサービスから認証認可機能を認証認可基盤として分離させこれを管理運用するため管理運用統合プレーンを含めるアーキテクチャを設計し実装を行った。

管理運用統合プレーンの評価については、構築した認証認可基盤および管理運用統合プレーンの性能評価を行う。性能評価としては共同研究内で活用されている環境、現在構築および整備を行っている KMD IdP (Identity Providor) のテスト環境での実際の管理運用を通した運用事例、運用者から意見による定性評価を行う。

また本研究の成果をもとに、管理運用統合プレーンを導入した認証認可基盤とその運用方法を標準化するための提言を行う。

本研究で提案する管理運用統合プレーンを採用したアーキテクチャは、様々な組織において容易に実装することができ、一定程度のシステム障害やアカウントの管理運用を行うことが可能となる。また実装した認証認可基盤、管理運用統合プレーン、その他ツール群についてはすべて公開しているためライセンスの範囲内で自由に利用することが可能となっている。これらにより現状の認証認可における問題を解決し実践的な認証認可基盤の実現に貢献する。

キーワード：

管理運用統合プレーン, セキュリティ, ネットワーク, アーキテクチャ, 認証認可基盤

慶應義塾大学大学院メディアデザイン研究科

加藤 大弥

Abstract of Doctoral Dissertation of Academic Year 2022

A Plane Architecture on Management and Operations of Authentication and Authorization Infrastructure

Category: Science / Engineering

Summary

In today's world where people use services on the Internet, the registration of Digital Identity and the authentication that accompanies it are indispensable elements. However, there is no end to the problems of leakage of personal information, unauthorized access, and service outages due to inadequate management and operation of authentication and authorization functions caused by the lack of an environment for managing and operating digital identities. In this study, this research consider this issue to be the difficulty in the management and operation of the system surrounding the authentication and authorization of existing methods, and attempt to resolve the current situation.

In order to solve the problem, this proposal defines a management and operational integration plane architecture in the authentication and authorisation infrastructure(AAI). A plane means a place where specific processing is carried out and a function, and the entire network is efficiently managed and operated by defining the layer that processes the function used to transfer data as the data plane and the layer that processes the functions of data transmission and connection control as the control plane. The objective of this research is to realise a practical authentication and authorisation infrastructure for operators by defining a management and operation integration plane in the authentication and authorisation infrastructure using this concept.

This research realise an integrated management and operation plane architecture, this study first investigated the system architecture of current services with authentication and authorisation and the factors that cause problems. Based on this, we designed and implemented an architecture that includes a management and operation integration plane in order to separate the authentication and authorisation functions from the service as an authentication and authorisation infrastructure and to manage and operate this.

For the evaluation of the management plane, this research evaluated the performance of the authentication and authorization infrastructure and management plane that this research had built. The performance evaluation will be based on the environment used in the joint research, operational examples through actual management and operation in the environment of KMD IdP (Identity Providor), which is currently being built and maintained, and qualitative evaluation based on the opinions of operators.

Based on the results of this research, this research will also make recommendations for standardizing a management plane to authentication and authorization mechanism and its operational methods.

The architecture proposed in this study, which a management and operation integration plane, can be easily implemented in various organisations and enables a certain degree of system failure and account management and operation. In addition, the implemented authentication and authorisation infrastructure, management and operation integration plane and other tools are all publicly available and can be used freely within the scope of the licence. These tools help to solve the problems of current certification and authorisation and contribute to the realisation of practical certification and authorisation infrastructures.

Keywords:

management and operation, security, network, system architecture, authentication and authorization infrastructure

Keio University Graduate School of Media Design

Daiya Kato

目　　次

図　目　次

表　目　次

第 1 章
序　　論

1.1. 本研究の背景

現在の我々の生活圏ではインターネットを活用したサービスを利活用することで豊かな生活を送るための様々な恩恵を得ており、日本においても、2021 年にデジタル庁 [1] が新たに設置され、より豊かなデジタル社会の形成が進められている。この社会基盤となりつつあるインターネット上の様々なサービスを利用するために個人が個人であるということを証明する認証と認証された個人に適切なリソースを提供する認可が必要不可欠である。インターネット上で認証認可を行うためには、Digital Identity が必要不可欠である。ではこの認証認可はインターネット上でどのように実現しているのであろうか。一般的には本人確認を行う認証、特定のポリシーをもとにサーバーから正しくサービスを提供するための認可、Digital Identity の管理を一連で行うことが可能なシステムを構築することで実現している。このシステムの名称は、時代や利用されている技術、プロトコルによって異なっており、例えば、認証認可システム、Identify Provider(IdP)、Identity and Access Management(IAM) と呼ばれている。本研究ではこのシステムのことを認証認可システムと呼ぶ。

この認証認可システムは世界中に数多に存在しており世界中の人々が何らかの形で必ずと言っていいほど活用している。最も一般的な活用事例としてはサービスを利用する際のユーザー登録とログインであろう。この際にサービスを利用するユーザーは数多くの属性情報を登録し、自身の Digital Identity として管理する。これを一般的に個人情報と呼び、日本においては、個人情報保護法によって明記されている通りその取扱いは非常に重要な位置づけとなっている。その理由とし

ては、個人情報保護法 第一条を引用する [2]。

> この法律は、デジタル社会の進展に伴い個人情報の利用が著しく拡大していることに鑑み、個人情報の適正な取扱いに関し、基本理念及び政府による基本方針の作成その他の個人情報の保護に関する施策の基本となる事項を定め、国及び地方公共団体の責務等を明らかにし、個人情報を取り扱う事業者及び行政機関等についてこれらの特性に応じて遵守すべき義務等を定めるとともに、個人情報保護委員会を設置することにより、行政機関等の事務及び事業の適正かつ円滑な運営を図り、並びに個人情報の適正かつ効果的な活用が新たな産業の創出並びに活力ある経済社会及び豊かな国民生活の実現に資するものであることその他の個人情報の有用性に配慮しつつ、個人の権利利益を保護することを目的とする。

つまり認証認可システムは、インターネット上でサービスを提供するためには不可欠な要素であり、個人を特定するための数多くの Digital Identity、個人情報を有している細心の注意を払い管理運用をする必要のある重要なシステムであることがわかる。

しかし残念なことに、認証認可システムが原因となる事故つまり一般的にいうところの情報漏洩が世界中で規模の大小にかかわらず数多く発生している。これは今日においても同様であり、情報処理推進機構の公開している情報セキュリティ 10 大脅威 2022 [3] の組織に対する脅威において情報漏洩の項目だけでも、2 位、4 位、5 位、10 位に挙げられている。代表的な事例としては、セブンペイ [4] やマリオットホテル傘下のスターウッドホテル [5] に挙げられるような大規模な情報漏洩である。

ではなぜこれらの大規模な情報漏洩が発生するのだろうか。情報漏洩の原因としてセブンペイの調査報告書を抜粋する [6]。主な原因として 3 つ挙げており、

1. 7pay に関わるシステム上の認証レベルが不足していた。
2. 7pay の開発体制が整っていなかった。
3. 7pay におけるシステムリスク管理体制不十分であった。

と報告している。規模の大きな組織においてもこのような状況に陥る原因としては、認証認可システムおよび認証認可の技術そのものが非常に難解であり、正しく構築、管理、運用を行うことが可能となる人材を含めた環境を整備することが困難であることが原因であると考えられる。では、正しく構築、管理、運用を実践できている組織はないのであろうかというとそうではない。例としては、Digital Identity を実践的に管理、運用しているだけではなく、収集している Digital Identity をソーシャルログインと呼ばれる機能を活用することが可能な認証認可システムとして成立させている組織もある。しかしこの認証認可システムの実践におけるノウハウや経験については、個人情報の漏洩につながるという観点から、広く一般的に公開されることは少なく、また各組織の独自の体制、手法によって確立されているため一般化されていないのが現状である。

1.2. 本研究の管理運用統合プレーンを導入した認証認可基盤

認証認可システムはインターネット上でサービスを実現するために今後も必要不可欠である。サービスの目的や用途により、パスワードで良いのか、多要素認証を用いるのかという認証と、ユーザーにどの程度の権限を与えるかという認可の認証認可ポリシーは異なる。また活用する認証認可システムについても、利用するソフトウェアやアカウント情報の保存方法などのシステムポリシーについても異なる。これらのポリシーは当然守られるべきである。また認証認可システムは、認証認可機能を継続的にサービスに提供する義務があり、認証認可システムの停止はサービスの停止を意味している。そのため認証認可システムは継続的に管理運用され続けるべきである。つまり認証認可システムは、組織の認証認可ポリシーとシステムポリシーを守り、継続的に管理運用が行えるアーキテクチャであるべきである。

現在の認証認可システムは、ポリシーの適用とシステムの管理運用を別の分野としてとらえているためトレードオフとなっている。IDaaS などのクラウド認証認可サービスを利用する場合では、サードパーティのポリシーに従わなければ利

用することができないため、組織のポリシーが適用できなければ利用できない。また組織のポリシーを守るために自組織の認証認可システムを利用する場合では、発生しているインシデントからもわかる通り、現在の認証認可システムのアーキテクチャでは両立を実現できていない。

これらのことから、組織の認証認可ポリシーとシステムポリシーを守りつつ、自組織で運用可能となる認証認可システムを実現するためには、従来の認証認可システムとは本質的に異なるアーキテクチャが必要であり、これが本研究で提案する認証認可基盤の管理運用統合プレーンアーキテクチャである。

1.3. 本研究の課題

本研究では、認証認可ポリシーとシステムポリシーを守り、自組織で運用可能となる認証認可システムを実現する管理運用統合プレーンを導入した認証認可基盤を提案し実現する。そのために従来手法で解決する課題を示す。

一つ目は、システムポリシーを自由に適用できるアーキテクチャとなっていないという課題である。組織のシステムポリシーに適したソフトウェアを自由に選択可能であり、組織の環境に対して影響が少なく、認証認可システムの専門的なノウハウを有していなくとも構築が可能となる認証認可システムであるべきである。そのためこれらを満たすシステムアーキテクチャであるべきである。

二つ目は、認証認可ポリシーを適切に運用可能となるインターフェースが定義されていないという課題である。従来では認証認可ポリシーを適応する手法が認証認可システムで利用しているソフトウェアやシステムアーキテクチャによって異なっているためポリシーの運用のためには専門的な知識が必要となっている。また自組織のシステムポリシーが変更された場合に、新しくノウハウを取得しなければならなず、認証認可ポリシーの適用が困難となっている。そのため、システムポリシーの変更に影響されず継続的に認証認可ポリシーを適応可能となるインターフェースが定義されるべきである。

三つ目は、認証認可システムを継続的に運用可能となるアーキテクチャとなっていない課題である。従来では認証認可システムを構成する各種要素は管理に必

要となる管理情報が定義されていないため、ソフトウェア毎に出力される管理情報が異なり、また専門的な知識がなければ運用に必要となる管理情報を選択できない。同様に管理手法についても定義化されているものはなく、利用しているソフトウェアの操作ノウハウが必要となり、システムポリシーが変更された場合には、再度新しくノウハウを取得しなければならない問題がある。そのため、システムポリシーの変更に影響されず継続的にシステムポリシーを適応し運用可能となるインターフェースが定義されるべきである。

1.4. 本研究の対象範囲

1.4.1 対象者、組織

本研究の対象者、組織は、あくまでも自組織内で認証認可システムを構築し運用する必要のある者および組織である。そのため、Identity as a Service と呼ばれるような既存の認証認可サービス、先にも挙げた、Google、Apple、Facebook、Amazon のようなソーシャルログインのような認証認可を機能として活用している者および組織については本研究の対象外である。

1.4.2 セキュリティの観点

本研究は目的は、認証認可システムを管理運用するための手助けとなる管理運用統合プレーンを定義することで実践的な認証認可基盤の実現であり、本研究の成果を活用することで、現在および今後認証認可システム関連で発生する情報漏洩、攻撃を完全に防ぐためのものではない。また同様にユーザーを起因とする問題の発生、例としてはユーザーが脆弱なパスワードを利用している、第三者にこれらの情報を伝えたことから発生する不正アクセス等については防ぐことはできないため研究の範囲外である。ただし実践的な認証認可基盤の実現を目的としているため不正アクセスの検知、ログの収集提供、ユーザーアクセスの停止などの管理運用の側面に関しては本研究の範囲内である。

1.4.3 運用者の選択とそれに伴う問題

認証認可基盤の構築、管理、運用を行う環境は多種多様である。これは運用者の考え方や組織のポリシーが異なるという点、利用しているソフトウェアによる環境の依存、資金面や人材の余剰など様々な要因があるためである。そのため本研究においてはその中でも、明らかに一般的な利用環境から逸脱している環境については本研究の対象とならない場合がある。

また本研究の認証認可基盤とそれに伴う管理運用統合プレーンは、利用する運用者に対して様々な要素を自由に選択可能な自由度のあるソフトウェアとなっている。もちろん本研究の目的でもある実践的に運用するための推奨すべき環境やアーキテクチャに関しては提示している。しかし先に述べた通り、認証認可基盤の構築、管理、運用を行う環境は多種多様でありすべての環境を網羅することは不可能であるため、この研究を活用する運用者の選択次第では本研究で想定していない問題や実装環境の破壊に伴うデータの消失などににについては対象外としている。

1.5. 本研究の貢献

本研究では自組織で認証認可システムを構築する必要がある環境、特に運用を行う人材や管理運用のノウハウが整っていない環境であっても、継続的に運用が可能となる認証認可基盤の実現を貢献とする。

また本研究を通して、認証認可基盤の管理運用についての標準化に向けた提言を行うことで、Internet and Operation Technology 領域や認証認可システム運用グループに対して情報共有と議論することで、将来的に認証認可基盤の管理運用のノウハウの蓄積と運用体制の底上げに貢献する。

1.6. 本論文の構成

本論文の構成としては、2 章でインターネット上のサービスにおける認証認可の必要性と機能を説明し、現状の認証認可システムの状況と問題点について示す。

3 章で従来の認証認可システムの課題と本研究を実現するために解決すべき課題について示す。4 章では、本研究における認証認可基盤と管理運用統合プレーンの設計を示し、5 章において 4 章で示したアーキテクチャをもとにした実際の実装について示す。6 章において実際の運用をもとに本研究で実装した認証認可基盤と管理運用統合プレーンの実証評価を示している。7 章に本研究の結果をもとに認証認可基盤の管理運用についての標準化に向けた提言を示している。最後に 8 章においてまとめと今後の課題について述べる。

第 2 章
認証認可システムの技術体系と現状

2.1. 認証認可における必要な要素と定義

本章では、本研究で取り扱う認証認可において必要となる各種要素とその定義について述べる。認証認可にまつわる用語の定義は各国、組織、団体、標準によって異り、また同じ意味として状況や条件においても大きく異なる。本研究ではあくまでもデジタル空間上における認証認可を取り扱うものとする。

2.1.1 認証認可の定義

認証認可の定義は、1994 年に Internet Engineering Task Force(以下 IETF) にて RFC1704 として N. Haller 氏、R. Atkinson 氏によって定義されている、“Authentication : The verification of the identity of the source of information.”、“Authorization : The verification of the identity of the source of information.” を引用する。日本語としての定義は、「認証は、利用者が登録している情報と一致するのか確認を行うこと。認可は、利用者が持つ権限に対応したリソースの利用許可を行うこと。」とする。

2.1.2 Identity の定義

る Identity の定義は、ISO/IEC 24760-1 [7] によって定義されている、“set of attributes related to an entity” を引用する。日本語としての定義は、「ある実体に関する属性情報の集合」とする。

2.1.3 Digital Identity の定義

Digital Identity の定義は、「Identity をデジタル空間上で利用可能である状態に表現したもの」とする。具体的な例としては、氏名、年齢、住所、パスワード、秘密鍵、証明書等の情報の集合を管理している Google や学認が挙げられる。本研究においては、この Digital Identity を、“ID” と表記する。

2.1.4 クレデンシャルの定義

Digital Identity の定義は、Digital Identity の中でもパスワード等の特に認証に用いられる機密な情報をクレデンシャルとする。

2.2. 今日の認証認可の役割と活用

本章では、本研究の目的である実践的な認証認可基盤の実現に対し、今日のインターネットにおける認証認可役割と活用事例について整理する。整理したのち認証認可の重要性と社会的な問題について述べる。

2.2.1 インターネットにおける認証認可

今日のインターネットにおける代表的な認証認可の役割について図 2.1 に示す。インターネットにおける主な認証認可の役割は、何かしらのサービスを利用する際に、利用者がサービスに対して本人であることを証明し、サービスが本人確認をしたのちに利用者に対して適切なリソースを提供することである。つまり認証認可の仕組みがなければ個人に対して適切なリソースを提供することができないということになる。

認証認可はあくまでも認証と認可の機能のみを示し、サービスを利用開始するつまりサービスに登録をする際の本人確認 (KYC:Know Your Customer) 提供するサービスのリソースの内容についてはこれに含まれていない。認証と認可の詳細については次に述べる。

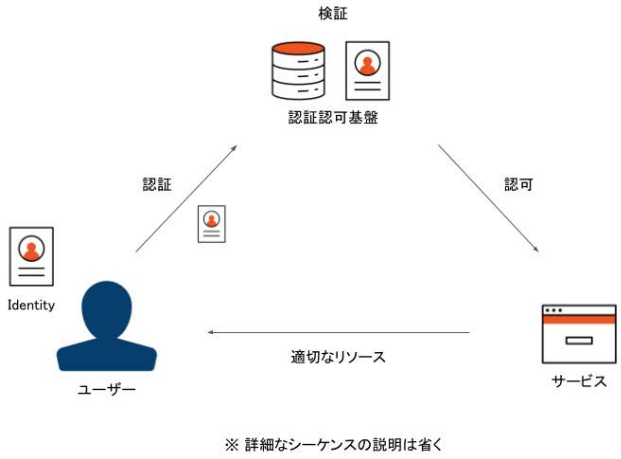

図 2.1 代表的な認証認可の役割

2.2.2 認証の役割と仕組み

認証の定義については、2.1.1 項に示している通り、利用者が登録している情報と一致するのか確認を行うことである。具体的な例としてよく挙げれるものとして図 2.2 に示しているマイナンバーカードを挙げる。

日本におけるマイナンバーカード [8] は、日本政府が本人確認を行ったうえで発行しているマイナンバーや顔写真、氏名等が記載されているカードである。本カードは日本政府が責任をもって発行しているという点で信頼できる Identity であると位置づけられているため、カードの顔写真と名前がカードの提示者と一致しているということを確認することが可能である。認証の役割は、この確認を行った結果一致しているのかどうかのみを示している。あくまでもそこに何かしらの権限や効力が生まれるということはない。これが認証である。

認証は大きく分けて 3 種類に分類することができ、一般的にはこの 3 種類の方法以外では本人確認を行うことはできないと現在は考えられている [9] [10]。その 3 種類は、

- 記憶 - Something You Know (Userid/Password、PIN)

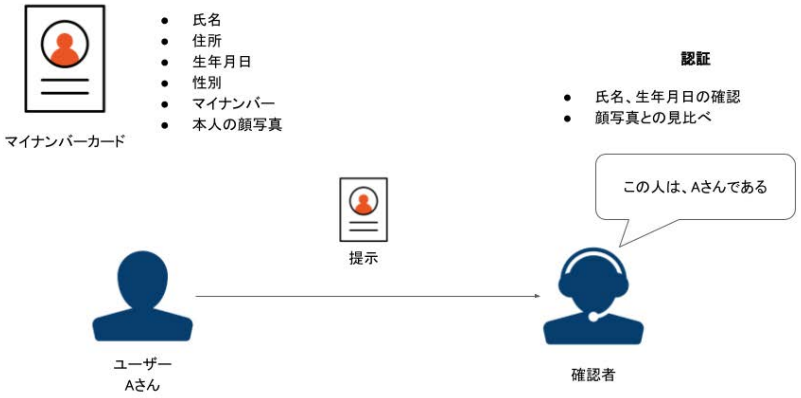

図 2.2 認証の役割

- 所持 - Something You Have (マイナンバーカード、証明書)
- バイオメトリクス情報 - Something You Are (指紋、静脈、顔)

であり、これは現実世界での Identity やインターネット上での Digital Identity に共通の事項となっている。つまり図 2.2 に示している例では、所持-Something You Have を活用した認証を行っていることになる。

2.2.3 認可の役割と仕組み

認可の定義については、2.1 章に示している通り、利用者が持つ権限に対応したリソースの利用許可を行うことである。具体的な例として図 2.3 に示す。

ここでは各種役所での例をもとに示す。2.2.2 項で示したマイナンバーカードで認証した後に、確認者は A さんに適した各種サービスを提供することができるようになる。このように各種役所、確認者が A さんに適した役所のサービスの利用の許可を与えることを認可と呼ぶ。その後 A さんが住民票の取得を求めた場合、この許可に準じて適切なサービス、ここでは A さんの住民票を提供するという一連の流れとなる。

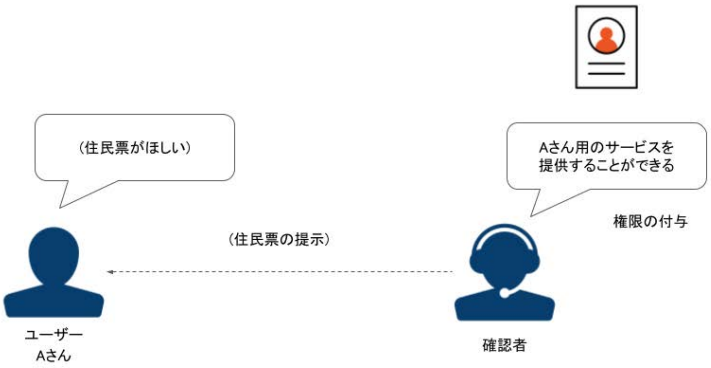

図 2.3 認可の役割

2.2.4 国際標準およびガイドライン

ここまでで述べた通り、認証認可はサービスの大小にかかわらずサービスを利用するユーザーに合わせたリソースを提供するために必要な仕組みであるため、提供するサービスに合わせて綿密に認証認可を設計する必要がある。この設計に少しでも不備があると、提示してきた情報と一致しないユーザーを認証してしまう、必要以上の権限をユーザーに認可してしまうという事故が発生する。これに 1 章で挙げたような、個人情報の漏洩や一般に公開していない機密情報の漏洩といった問題を起こし重大インシデントとなる可能性が十分にある [4]。

そこで認証認可にまつわる一連のアーキテクチャや運用についての指南として、認証認可におけるポリシーや組織のアーキテクチャについての様々なガイドラインが存在している。このガイドラインは発行した組織や国に合う形式でまとめられており国際的な決定されている標準や法的な根拠はないことが多い。これは一意に提供するサービスや状況が同様なものはなく自組織のポリシーに適合している設計を行っているためである。ただし、NIST SP800-63-3 [10] を代表とする一部のガイドラインは発行元の信頼に伴って世界的に参考にされているガイドラインとなっており、認証認可機能を運用していくために利用されることが多い。

ここでは、NIST SP800-63 について詳細を示し、認証認可におけるガイドラインの現状について述べる。NIST SP800-63 とは、

> These guidelines provide technical requirements for federal agencies implementing digital identity services and are not intended to constrain the development or use of standards outside of this purpose. The guidelines cover identity proofing and authentication of users (such as employees, contractors, or private individuals) interacting with government IT systems over open networks. ” [10]

と示されている通り、米連邦政府機関を対象とした認証認可のあり方を定義したものである。このガイドラインは 3 種類でポリシーを提示しており、“Enrollment and Identity Proofing”、“Authentication and Lifecycle Management”、“Federation and Assertions” となっている。この文章はあくまでもアメリカ政府におけるポリシーを定めたものであり、技術的な仕様や実際の認証認可システムについては触れていない。そのためアメリカ政府と Digital Identity を利用したやり取りを行う場合、各々でこのポリシーを認証認可システムに技術的に落とし込んでいく必要がある。

SP800-63-3 を例に挙げた通り認証認可におけるポリシーやアーキテクチャについてのガイドラインは数多く存在しており、各国の政府だけではなくこれらのガイドラインを参考にして大小さまざまな組織、企業に合う形で技術的に適応させることで実践的な認証認可基盤やシステム全体のアーキテクチャを構築することが可能である。しかし現状では認証認可基盤にこれらのガイドラインが反映されているとは言えない。その実情を次節で示す。

2.2.5 ガイドラインの社会への影響と反映状況

ここでは再び、SP800-63-3 をもとに社会への影響と実際にどの程度これらのガイドラインが反映されているのかを日本におけるサービスを事例に示す。

まずパスワードの定期変更について例を示す。SP800-63-3B にはパスワードの定期変更については不要である旨の記載がされている。これまでは一般的にパス

ワードを定期的に変更することで強固なパスワードを都度生成した方が強度が高いとされてきたが、Weir Matt [11] らの調査結果から、“password0”,“password1” など潜在的に脆弱なパスワードが生成される可能性が高いことが示され、定期変更は必要ないという判断となっている。これにより、2018 年に総務省は企業組織に対して、「これまでは、パスワードの定期的な変更が推奨されていましたが、2017 年に、米国国立標準技術研究所（NIST）からガイドラインとして、サービスを提供する側がパスワードの定期的な変更を要求すべきではない旨が示されたところです。また、日本においても、内閣サイバーセキュリティセンター（NISC）から、パスワードを定期変更する必要はなく、流出時に速やかに変更する旨が示されています」[12] という内容に変更されている。しかし方針が変更されてから 4 年が経過するににもかかわらず、図 2.4 を例に示すような日本におけるサービスではいまだにパスワードの定期変更を促すものも多く存在している[1]。

次に秘密の質問について例を示す。秘密の質問とは、認証の一環として、“初めて飼ったペットの名前は”、“母親の旧姓は” などの質問を要求するものである。これについては SP800-63-3B において “SHALL NOT”、つまりしてはならないという見解を示しており、ペットの名前としては、“タマ”、“ポチ”、苗字としては、“伊藤”、“佐藤” など統計的に突破しやすい条件が生まれてしまうことに由来している。これらはログイン時の追加情報やパスワード紛失時のアカウントリカバリーの際に目にすることが多いのではないだろうか。

これらで示した通り、ガイドライン上では利用することが推奨されていない要素であっても、反映されていないサービスが存在していることを示した。

ではなぜこのようにガイドラインをサービスに反映することが困難であるのかを考察する。一つは現在運用しているサービスと認証認可機能、それに伴うポリシーに関してである。例えばワンタイムパスワード新しい新しい認証要素を導入した場合、認証を行うエンドユーザーに対して教育や訓練が必要となり、新技術のベネフィットとリスクを踏まえて判断する場合である。これは自組織で判断を行う環境が整っている結果と考察でき本研究の対象とはしない。次に最新の認証

1 野村證券 当社からのお知らせ パスワードの定期的な変更をお願いいたします https://www.nomura.co.jp/onlineservice/news/customer/2021/l065v40000042mfx.html

図 2.4 パスワードの定期変更を促す例

ガイドラインに準拠した運用は専門家がいないと困難である点である。これは認証認可の技術体系とプロトコルの複雑さから認証認可基盤の管理運用が困難であり、この複雑性がインシデントを発生させている原因となっていると考える。そのため複雑性を解決するために、運用の専門家ではない新規運用者が容易に利用できる認証認可基盤を提供する必要がある。2.3 節にこの複雑性について示す。

2.3. サービスと認証認可の技術体系の変容

本章では、2.2.4 項、2.2.5 項で挙げたガイドラインを実際のサービスに反映できていない現状を、現状のサービスのアーキテクチャと認証認可の技術体系をもとに考察し、その課題について述べる。認証認可の歴史は UNIX の複数ユーザーでのマルチタスクを実現するためのマルチユーザー機能からはじまるが、本研究では Web サービスにおける認証認可の変容について述べる。

2.3.1 サービスと認証認可機能一体型のアーキテクチャ

まずサービスと認証認可機能についての原始的なアーキテクチャを図 2.5 に示す。これは基本的なアーキテクチャであると同時に現在も広く一般的に利用されているアーキテクチャとなっている。主な構成要素として、サービスのコンテンツ、Digital Identity を保存しておくデータベースである。このアーキテクチャは 1 つのサービス内にコンテンツと認証認可機能が共存しており、コンテンツと認証認可機能の依存関係が密になっている。本アーキテクチャの基本的な流れとしては、ユーザーが提示した Digital Identity を事前にデータベースに保存している Digital Identity と検証を行った後、検証結果をコンテンツに渡すという仕組みである。

この構成の問題としては、 サービス内にすべてのシステム、ソフトウェアとデータを含んでいることである。これらから発生する課題としては、、

- コンテンツリソース、クレデンシャル情報、Digital Identity が一つのデータベースに保管されるため同様の管理運用となること
- ソフトウェア同士が密な依存関係にあるため新規認証認可技術の導入が困難であること
- 複数のサービスを構築した場合、サービスに保存されているクレデンシャル情報が増加してしまうこと

の 3 つが課題となる。まず 1 つ目については、データベースに重要度の異なるデータが共存しているという問題に起因する。図 2.6 に示すように、ポリシーや組織

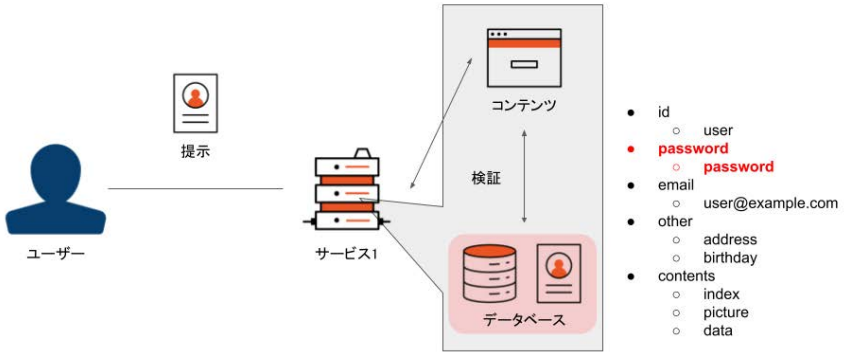

図 2.5　サービスと認証認可の原始的なアーキテクチャ

によって異なるが一般的にデータは、外に漏れてはいけないパスワードのようなクレデンシャル情報や、漏れても迷惑メール程度が送られてくる程度の email アドレス、そもそも公開しているので漏洩しても大きな影響がないコンテンツの公開リソースのように、その特性の違いからから管理運用手法が大きく異なる。図 2.5 では、これらすべての情報が同一のデータベースに保存されているため、重要度の高い情報の管理運用手法に合わせなければならない。これは研究室内の wiki から企業の大規模サービスまで例外なく本来であれば管理運用を行う必要があるが、管理運用を行う環境の規模や人材によってその難易度は当然差異が生まれる。また難易度が上がるほど管理運用を行うべき要素が増えるため細かいミスが発生する場合もある [13]。

2 つ目は、新規の認証認可技術への対応についてである。NIST SP800-63-3 [10] や 2.2.4 項、2.2.5 項でも述べたように、認証認可技術やポリシーは突如として大きな変更を行わなければならない状況となる。また次節で詳細は述べるが、ワンタイムパスワード、生体認証などの認証技術や SAML、OpenID Connect などのプロトコルが新しく生まれ、これらがデファクトスタンダードとなった場合の社会的風潮による対応を求められる。警察庁の令和 3 年度の不正アクセス行為対策等の実態調査 [14] にもある通り、多要素認証の導入は 21.4%、リスクベース認証

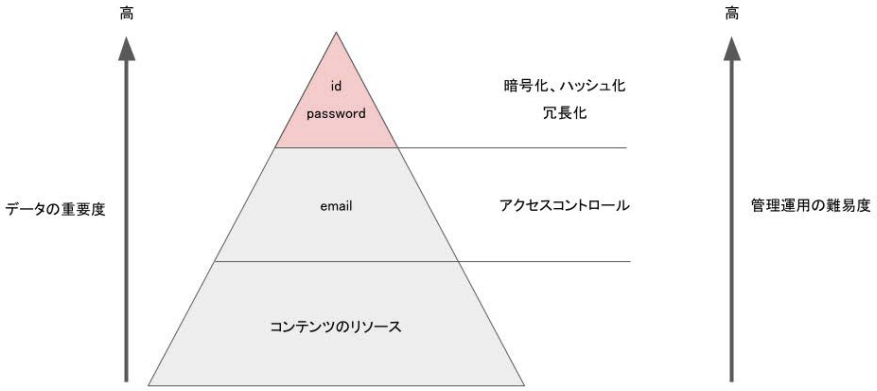

図 2.6　データの重要度よ管理運用の関係性

の導入は1.1%となっており、新しい技術の導入が困難であることが伺える。この理由としてはシステムが対応していないことや、新しい認証要素に対してエンドユーザーが習熟していないことへのユーザビリティとの兼ね合いが挙げられるが、様々な認証要素が活用できる設計となっている必要がある。

3つ目は、管理運用しなければならないクレデンシャル情報の増加である。新規サービスを組織で追加する場合、本アーキテクチャの場合ユーザー数によって変動はあるものの図2.8で示す通り、単純増加でサービス数の倍数のクレデンシャル情報を管理運用することになる。そのため最終的に1つ目の課題がしてしまうとともに情報漏洩の可能性を上げてしまうことになる。

このようにこのアーキテクチャにはコンテンツや認証認可機能の管理運用上の課題があることを示した。次節ではこれらの課題を解決を図るためのアーキテクチャについて述べる。

2.3.2 サービスと認証認可基盤に分かれたアーキテクチャ

これら問題の根本をコンテンツと認証認可機能が共存していることにあると捉え、これらを分離し別々の機能として管理運用するアーキテクチャについて述べ

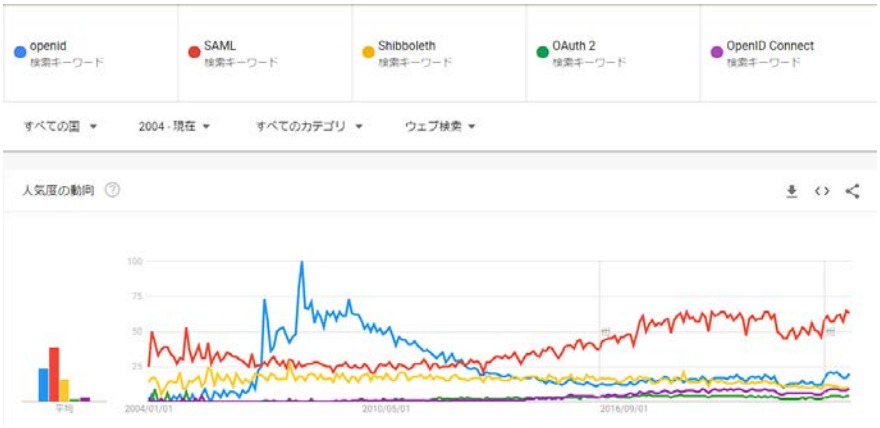

図 2.7　過去 15 年間の Google の検索トレンドの変化

る。このアーキテクチャはサービスに対する、ID 連携、Identity provider(IdP) と一般的に呼ばれており、図 2.9 に示す通りコンテンツを管理するシステムと認証認可を行う認証認可基盤に分離したものになっている。この手法を用いることで、重要なデータであるクレデンシャル情報の分離、認証認可に特化したシステムの導入と管理運用による新規技術導入コストの低減 (図 2.9)、サービスの増加に伴うクレデンシャル情報の増加 (図 2.8) の 3 つの課題を解決することが可能になる。

このアーキテクチャの具体的な例を 2 例挙げる。1 つ目の例は図 2.10 に示すソーシャルログインである。ソーシャルログインとは、Google アカウント [15]、Twitter [16] アカウントなどすでに登録しているサービスのアカウントを活用して対象サービスに対して認可を行うものである。2 つ目は図 2.11 に示す学認 [17] や Azure Active Directory [18] を代表とする認証連携機能および Single Sign On(SSO) 機能である。これも同様に認証認可基盤に事前に登録しているアカウント情報を利用したいサービスに提供することで認証認可を行うものである。つまり認証認可基盤としてこれらを活用することにより、認証認可の課題である急速に変容するガイドラインやポリシーの反映を行うことが可能になるはずである。しかしこれらを利用するということは認証認可機能の実装、運用、管理を他組織に依存し

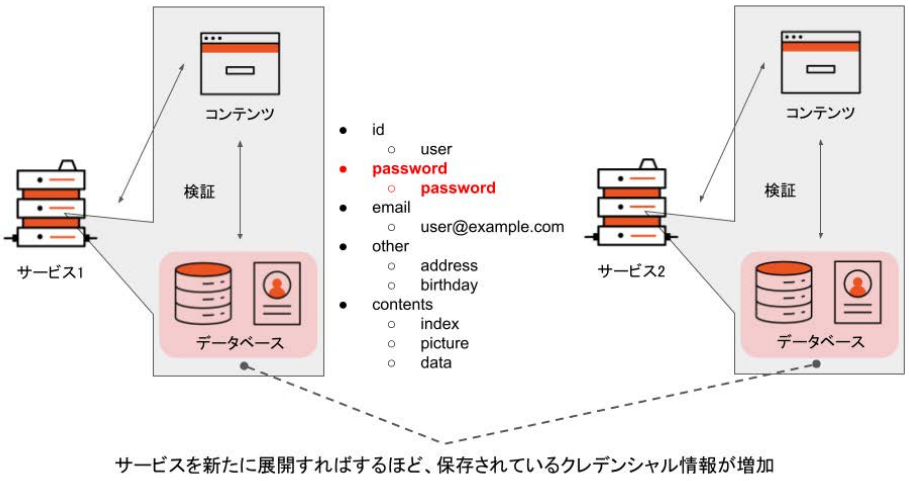

図 2.8 サービス増加に伴うクレデンシャル情報の増加

てしまうため、自組織では何もできなくなる状況に陥るという課題も見受けられる。これらをもとに 2.2.5 項や本章で示した調査結果のようになぜ対応が芳しくないのかを考える。

2.4. 認証認可技術の標準とプロトコル

認証認可技術の標準とプロトコルについてまとめ、技術仕様から見た課題について述べる。認証認可技術標準やプロトコルは数多く定義されている。ここではその中から現在一般的に用いられている、SAML と OpenID Connect を比較しながら認証認可技術の抱える課題について述べる。

SAML と OpenID Connect は、図 2.9 に示す認証認可機能をサービスに提供するための技術標準である。用途としての大きな違いは、SAML が社内ネットワーク向け、OpenID Connect は外部の一般サービス向けに認証認可機能を提供することにあり、成り立ちとしては、2005 年に SAML が標準化された後に、Web サービスやモバイルアプリケーションの普及に伴った Digital Identity 管理の増加による新しい ID 管理の必要性から、2014 年に OpenID Connect が標準化されている。

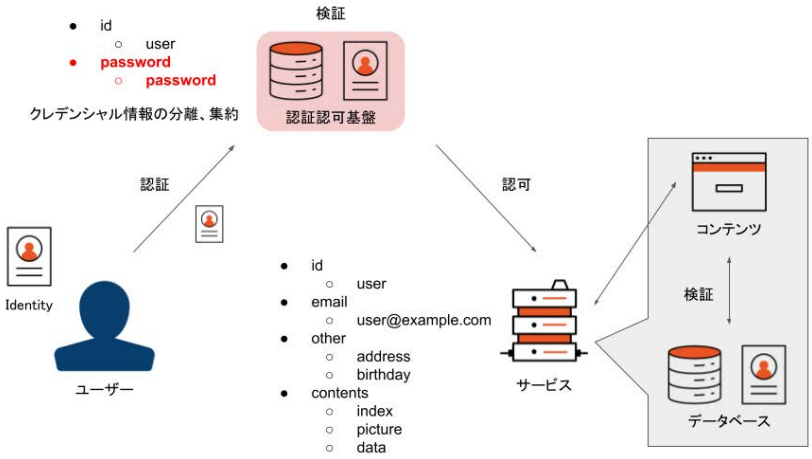

図 2.9 サービスと認証認可基盤に分かれたアーキテクチャ

それぞれの技術仕様についての詳細を示す。

SAML

SAML(Security Assertion Markup Language) は、2005 年に OASIS にて標準化 [19] された ID 連携を行うためのマークアップ言語である。認証認可におけるアサーションと呼ばれる情報を XML で定義しており、メッセージの送受信には SOAP を活用している。現在の最新版は SAML2.0 で最終更新は 2008 年である。SAML のドキュメントは、約 1.5 万単語となっている。

- Security Assertion Markup Language (SAML) V2.0 Technical Overview [19]

OpenID Connect

OpenID Connect は、2014 年に OpenID Foundation にて標準化 [20] された Web アプリケーション向けの認証を行うための技術である。この技術は OAuth2.0 の認可技術を拡張することで認証認可を実現している。認証認可におけるセキュリティ

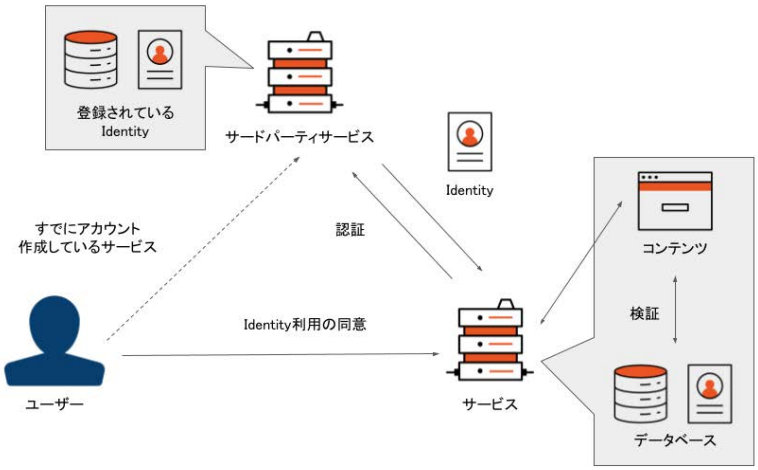

図 2.10 ソーシャルログインの例

情報を JSON 形式で定義しており、メッセージの送受信には JWT 形式で HTTP を活用している。OpenID Connect のドキュメントは publish(draft は除く) されているものだけを対象にすると、約 6 万単語となっている。

- OpenID Connect Core 1.0 [21]
- OpenID Connect Discovery 1.0 [22]
- OpenID Connect Dynamic Client Registration [23]
- OAuth 2.0 Multiple Response Type Encoding Practices [24]
- OAuth 2.0 Form Post Response Mode [25]
- OpenID Connect RP-Initiated Logout 1.0 [26]
- OpenID Connect Session Management 1.0 [27]
- OpenID Connect Front-Channel Logout 1.0 [28]
- OpenID Connect Back-Channel Logout 1.0 [29]

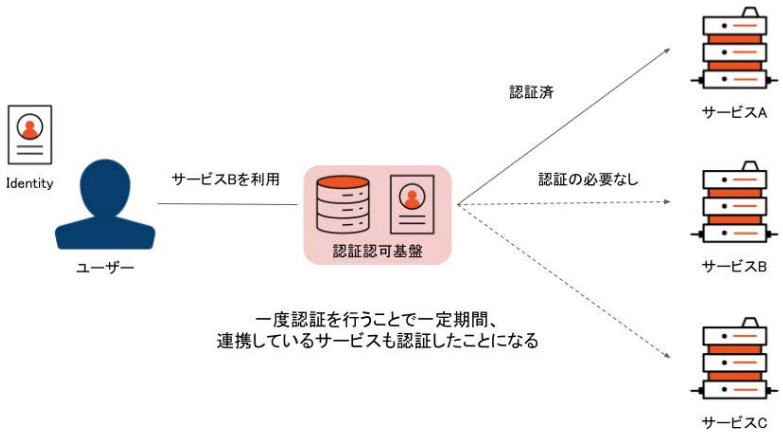

図 2.11 SSO の例

また標準とは別に、実装者向けのガイドラインが 2 種類用意されている。

- OpenID Connect Basic Client Implementer's Guide 1.0 [30]
- OpenID Connect Implicit Client Implementer's Guide 1.0 [31]

次に OAuth2.0 の関連標準について publish(draft は除く) されているドキュメントは、約 15 万単語となっている。また、OAuth2.0 関連標準は IETF Security oauth グループ [19] として現在も活発に数多くの標準化を行っている。

- RFC 6749 The OAuth 2.0 Authorization Framework [32]
- RFC 6750 The OAuth 2.0 Authorization Framework: Bearer Token Usage [33]
- RFC 6755 An IETF URN Sub-Namespace for OAuth [34]
- RFC 6819 OAuth 2.0 Threat Model and Security Considerations [35]
- RFC 7009 OAuth 2.0 Token Revocation [36]

- RFC 7519 JSON Web Token (JWT) [37]
- RFC 7521 Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants [38]
- RFC 7522 Security Assertion Markup Language (SAML) 2.0 Profile for OAuth 2.0 Client Authentication and Authorization Grants [39]
- RFC 7523 JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants [40]
- RFC 7591 OAuth 2.0 Dynamic Client Registration Protocol [41]
- RFC 7592 OAuth 2.0 Dynamic Client Registration Management Protocol [42]
- RFC 7636 Proof Key for Code Exchange by OAuth Public Clients [43]
- RFC 7662 OAuth 2.0 Token Introspection [44]
- RFC 7800 Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs) [45]
- RFC 8176 Authentication Method Reference Values [46]
- RFC 8252 OAuth 2.0 for Native Apps [47]
- RFC 8414 OAuth 2.0 Authorization Server Metadata [48]
- RFC 8628 OAuth 2.0 Device Authorization Grant [49]
- RFC 8693 OAuth 2.0 Token Exchange [50]
- RFC 8705 OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens [51]
- RFC 8707 Resource Indicators for OAuth 2.0 [52]

- RFC 8725 JSON Web Token Best Current Practices [53]
- RFC 9068 JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens [54]
- RFC 9101 The OAuth 2.0 Authorization Framework: JWT-Secured Authorization Request (JAR) [55]
- RFC 9126 OAuth 2.0 Pushed Authorization Requests [56]
- RFC 9207 OAuth 2.0 Authorization Server Issuer Identification [57]
- RFC 9278 JWK Thumbprint URI [58]

以上、OpenID Connect は、総単語数 21 万単語の仕様で構成されている。

2.5. 認証認可システム活用への課題

このように認証認可機能を担っている技術標準は、とても膨大なドキュメントから見てもわかる通り非常に巨大である。当然これらのドキュメントを読み尚且つ実際に技術を習熟することで認証認可技術を実装することが可能となり、また様々なサービスに対応するための認証認可システムを実装するには、少なくとも SAML、OpenID Connect 両方の習熟だけではなく、Digital Identity やクレデンシャル情報の管理方法、サービスへの対応についても習熟する必要がある。ではこれらを習熟するためにはどのようすればよいのであろうか。昨今の技術習得方法として用いられているのはベストプラクティスという手法である。ベストプラクティスとは専門的な技術を有している企業や組織、個人などが技術ブログや技術記事を投稿することで技術取得に必要なノウハウを公開しているというものである。しかし認証認可技術についてのベストプラクティスは少ないにも関わらず、考慮しなければならない要素が膨大でありこれらすべてを理解する必要がある。またサービス全体のアーキテクチャの違いによりノウハウが異なるため、“最終的には自身で考えてください。“という内容のものが必然的に多くなってしまうのが実情である [59]。例として、世界的な認証認可クラウド基盤を提供している Auth0 [60] のベストプ

ラクティスの項目内においても、“Storing tokens in browser local storage provides persistence across page refreshes and browser tabs, however if an attacker can achieve running JavaScript in the SPA using a cross-site scripting (XSS) attack, they can retrieve the tokens stored in local storage. A vulnerability leading to a successful XSS attack can be either in the SPA source code or in any third-party JavaScript code (such as bootstrap, jQuery, or Google Analytics) included in the SPA.“という注意項目があるように組織ごとのポリシーやアーキテクチャに合わせて自身で考えなければならない部分がどうしてもでてきてしまう。

ここまでで認証認可技術の習得は困難であることがわかったが、本来の目的は認証認可技術の習得ではなく、サービスの一部として認証認可システムを実践的に導入し活用することである。では仮に認証認可に関係する技術を習得し、ベストプラクティスから活用方法を身につければ認証認可基盤を使いこなすことができるかというとこれは違う。実践的に認証認可システムを利用するためには管理運用する技能がなければならない。

では、世の中のサービスではいったいどのようにして図 2.9 のアーキテクチャを実現しているのだろうか。

2.6. 認証認可基盤と IDaaS

ここまでで認証認可基盤を活用していくには、巨大な技術体系の習得だけではなく、認証認可基盤をシステムとしてどのように管理運用するべきかという技能、ノウハウが必要であることが課題であることを示した。これらの課題を解決しサービスを図 2.9 のアーキテクチャを実現するために、IDaaS(Identity as a Service) と呼ばれるクラウドサービスが 2017 年頃より台頭している。

IDaaS とは、これまで述べてきた課題を解決することを目的とした認証認可基盤を提供するクラウドサービスの通称であり、代表的なものとして、Okta [61]、Auth0 [60]、Azure Active Directory [18] があり、その市場規模は ITR の調査 [62] によると、2020 年度の売上金額だけで 51 億円、2025 年度まで二桁%で成長し続けると予想している。また世界最大の IDaaS 企業である Okta の 2022 年度売上高は

13 億ドル前年度比 64%増となっており、IDaaS の社会への必要性が伺える。IDaaS の基本的な機能は、フルマネージされた認証認可基盤の提供である。認証認可技術の実装や提供だけではなく、Digital Identity 管理運用、認証認可基盤自体の管理運用、不正アクセスの検知などのセキュリティ機能をサービスとして簡単に利用することが可能である。つまり IDaaS を活用することでこれまで述べてきた認証認可基盤を実践的に活用するために発生する課題はほぼ解決することが可能である。ではなぜ各組織のサービスにおいて IDaaS が利用されていないのであろうか。活用されていない理由としては、クラウド上にパスワードを預けられないような組織のポリシーやデータが万が一消える [63]・障害発生時に代替する方法がないという理由があげられる。

このように様々なニーズに対応して図 2.9 を実現するためには、多様なサービス形態があってこそ高い可用性が得られるため IDaaS、自組織での認証認可基盤の管理運用のどちらにおいても必要性があることがわかる。しかし残念なことに、自身で運用している認証認可基盤では、1 章で示した通り、管理運用を原因とする大きなインシデントの発生が後を絶たないのが現状である。

第 3 章
従来の認証認可システムの事例と課題

3.1. 認証認可システムを活用する環境

ここまでを踏まえて現状の認証認可を取り巻く現状とその環境について述べる。まず認証認可を利用している環境をアカウントの管理 (自組織か他組織か)、認証認可機能の提供 (自組織か他組織か)、認証認可機能とサービスとの独立性 (独立か非独立か) の 3 軸、計 8 種類の環境として考える。

1. アカウントの管理が自組織、認証認可機能の提供が自組織、認証認可機能とサービスが独立
2. アカウントの管理が自組織、認証認可機能の提供が自組織、認証認可機能とサービスが非独立
3. アカウントの管理が自組織、認証認可機能の提供が他組織、認証認可機能とサービスが独立
4. アカウントの管理が自組織、認証認可機能の提供が他組織、認証認可機能とサービスが非独立
5. アカウントの管理が他組織、認証認可機能の提供が自組織、認証認可機能とサービスが独立
6. アカウントの管理が他組織、認証認可機能の提供が自組織、認証認可機能とサービスが非独立

7. アカウントの管理が他組織、認証認可機能の提供が他組織、認証認可機能とサービスが独立

8. アカウントの管理が他組織、認証認可機能の提供が他組織、認証認可機能とサービスが非独立

この 8 つの環境の中から存在しにくい環境を考えると 4、8 については認証認可機能の提供者と認証認可機能とサービスの独立性が一致していないため矛盾している。また 3 のように自組織でアカウントを管理しており、外部の認証認可機能を利用することについてもネットワークアーキテクチャ上考えにくい。

次に残った 5 種類の環境とそのユースケースについて考察する。まずアカウントの管理者が他組織にある場合については、すでにアカウントを有しているサードパーティのサービスを活用するか IDaaS を活用してるユースケースが考えられる。次にアカウントの管理者と認証認可機能の提供者が自組織にあり、認証認可機能とサービスが独立場合についてはすでに自組織において認証認可基盤を構築引用している段階であると考えらる。最後にアカウントの管理者が自組織、認証認可機能の提供者が自組織、認証認可機能とサービスが非独立の場合は、自身でサービス毎に認証認可機能を導入し、アカウントをサービス毎に管理運用してるユースケースである。それぞれのユースケースを整理すると、

1. サードパーティに既に登録されているアカウントを活用する (7)

2. IDaaS を活用して自身でアカウントを管理運用する (5、6、7)

3. 自身で認証認可基盤を構築し、アカウントを管理運用する (1)

4. 自身でサービス毎に認証認可機能を導入し、アカウントをサービス毎に管理運用する (2)

となる。

このユースケースをもとに本研究における対象となる環境について述べる。まず、1 については、ユーザーが指定のサードパーティのアカウントを有していることを前提として、自身でアカウント情報を保持することなくサードパーティのア

カウント情報を活用している環境である。これはサービスを利用する人がGoogleやFacebook等に登録していれば、誰でも自身のサービスを利用してもよいというオープンなポリシーに基づく場合である。世界中に広く誰にでもサービスを展開している環境でよく見られる。2については、自組織でアカウントを新たに作成してもらう必要があり、認証認可とアカウント管理をクラウドサービスを活用して行う環境である。2.6節でも述べた通り、クラウドサービスを活用することが可能な人材、資金を有しており、アカウント管理のポリシーがクラウドサービス利用に適している場合である。3については、自組織でアカウントを新たに作成してもらう必要があるが、何らかの理由で2を利用できないため自組織で認証認可基盤を構築し管理運用する環境である。1章でも示した主に大規模な組織や大規模なサービスを展開している場合に見受けられる。4については、自組織でアカウントを新たに作成してもらう必要があるが、2を活用することができず、尚且つ3を自組織で運用する環境が整っていない場合である。認証認可に対しての知識の習得を十分に行うことができていない場合を想定している。

次にそれぞれの環境における認証認可とアカウントの管理運用の課題について述べる。1については、認証認可およびアカウントの管理運用のすべてをサードパーティに一任しているため、これらに関する管理運用面での課題は特にない。ただしすべてを一任していることからサードパーティ終了によるベンダーロックインが課題となっている。2については、認証認可基盤の管理運用および不正ログインの検知等のセキュリティ面でのアカウントマネジメントをIDaaS側がすべて担保するため、これらについての課題は特にない。ただし異常を検知した後のアカウントに対する運用やアカウントの追加や削除などのアカウントの管理運用を行う必要があるが、これらについてもサービスとして提供されているため比較的容易に実施することが可能である。課題としては1と同様に、IDaaSが終了した際のベンダーロックインや個人情報を国外に保存しないもしくは自組織で管理するというポリシーがある場合利用することができないことが挙げられる。1、2のセキュリティ面に関しては、サードパーティやIDaaSのDigital Identityを管理する環境が整っており、尚且つノウハウや経験も多く管理運用能力が高いため大規模な個人情報の漏洩などのインシデントの発生も少ない。

ここまでで、自組織でアカウントを作成する必要があり、ベンダーロックインを防ぎ、ポリシーとして Digital Identity を自組織で管理運用する必要がある場合に 3、4 の環境を選択することになる。まず 3 については、認証認可基盤を構築し管理運用、アカウントマネジメントをすべて自組織で行うための環境が整っているのかが課題となる。もちろん IDaaS を提供している組織のように自組織に認証認可技術の専門家、セキュリティマネジメントを行うことができる人材を十分に有している場合にはこの課題が解決可能である。また人材を育成することでも同様である。しかし 2.4 節で挙げた通り人材育成は非常に困難である。しかしこのような人材が十分ではなくとも、インターネット上からベストプラクティスや様々な組織の知見を用いてサービスを開発することは珍しくはない。しかし 2.5 節で挙げた通り認証認可については、セキュリティ面からこれらの知見が公開されることは少ない。そのため 1 章で示した、“十分な人材も管理運用のノウハウもない“、“独自の脆弱な認証認可基盤を構築してしまう “という環境が整っていない課題発生し尚且つ解決のためのベストプラクティスも用意されていない現状となっている。4 については、3 と同様の課題を根本的に抱えており、かつ 2.3.1 節で挙げた、現在の管理するべき Digital Identity を増加させないというセキュリティの潮流を取り入れることができていないという課題がある。

3.2. 本研究のターゲット

3.1 節より、本研究のターゲットについて述べる。本研究のターゲットは自組織で認証認可システムを管理運用する必要があるが、十分な人材も管理運用のノウハウがない、もしくはすでに独自の認証認可システムを構築している組織 (3.1 節の 3)、認証認可を行う必要があるがこれから認証認可システムを導入する必要がある組織 (3.1 節の 4) を対象とする。

3.3. 従来の認証認可システムのアーキテクチャ

従来の認証認可システムのアーキテクチャを例に、本研究のモチベーションである、各組織の認証認可ポリシーとシステムポリシーを守り、自組織で運用可能となる認証認可システムが実現されていないことを示す。ここでいう認証認可ポリシーでは、自組織で定義した認証認可に関するポリシーが適用できること、システムポリシーでは、自組織の環境やポリシーに合わせて自由に認証認可システムの構成要素やソフトウェアを選択できることである。

学術認証フェデレーションである学認 [17] は、全国の大学等の教育機関と NII が連携して、学術 e-リソースを提供するために相互に認証連携を実現する認証認可システムである。これにより各大学間での認証連携と学内サービスに対するシングルサインオンを実現している。学認を利用するためには、学認フェデレーションに参加すると共に自組織において学認をサポートするための認証認可システムを構築、運用をする必要があり、各大学に対する認証認可システム構築、運用マニュアルが提供されているため、これについて考察する。本マニュアルは、“技術ガイド “、“実習セミナー “、“SP 接続情報 “、“学認申請システム補足マニュアル “で構成されており、“技術ガイド “で認証認可機能を提供する Identity Provider(IdP) について、“実習セミナー “では Identity Provider(IdP) の管理、運用について説明しておりこれを取り上げる。まず学認を利用するためには、動作確認されており各組織に対して冪等性を担保するために、自組織のシステムポリシーに関係なく “CentOS 7“、“OpenLDAP“、“Jetty 9.4“、“Java 11“を必ず利用しなければならない。また認証認可システム全体のアーキテクチャについては提示されておらず、そのため大学内で展開されている認証認可システムを要素を満たすアーキテクチャを構成する必要がある。認証認可システムの運用については、必要となる設定ファイルとその概要については記されているものの具体的な運用手法については明言していない。また 2015 年に実施した学認アンケート [64] では、“本稿で述べた「理想的な IdP 運用」の体制は，ただちに導入できるような簡単なものではありません．その多くの要素は組織的な整備を要するもので，また IdP 運用の各機関の内情にあわせてカスタマイズする必要もあるでしょう．“と結論付けている。これらのことから学認の認証認可システムアーキテクチャにおいては、自組織

のシステムポリシーを適応することができない。また認証認可システムの管理運用手法については触れておらず、自組織で運用可能であることを示してはいない。

厚生労働省が取り組んでいる医療機関向け認証認可システムの調査研究 [65] では、“シームレスな健康情報活用基盤実証事業 “として実際に能登北部の医療機関において構築された認証認可システムの仕様書を公開している。本アーキテクチャでは認証認可システムで必要となっている機能と構成要素が洗い出されているため、検証は必要となるものの自組織にシステムポリシーに合わせたソフトウェアを用いて認証認可システムを構築可能となっている。認証認可においては、能登北部の医療機関に合わせたポリシーとなっていおり自由に認証認可のポリシーを適用することができないアーキテクチャとなっている。このことから多様な認証認可を実現するために、2020 年に NRI セキュアテクノロジーズ株式会社により “認証認可の調査研究 “が行われ [66]、本調査により医療機関向けの様々な認証認可技術やプロトコルの利用を実現した認証認可システムのアーキテクチャが設計され検証されている。しかし認証認可システムの運用については、“本調査研究では、処理性能に関する実機検証、基盤運用など運用に関する実機検証は想定しない。“と運用を別分野と捉えており、本アーキテクチャにおいてもポリシーの適用とシステムの管理運用の両方を実現できていない。

EU(欧州連合) が取り組んでいる EOSC-hub AAI [67] では、EU 内の学術組織に対して研究インフラストラクチャを提供するための認証フェデレーションを提供している。EU 内の各研究組織の IdP(Identify Provider) と EOSC-hub で ID 連携するために、認証認可システムのアーキテクチャと構成を公開している。本アーキテクチャでは、EOSC-hub で様々な認証プロトコルに対応することで各組織で自由な認証ポリシーを利用可能としている。認可ポリシーについても EOSC-hub が定義している範囲内で自由に選択することを実現している。また先の二例とは異なり、システムの管理を行うためのモニタリング基盤についてもアーキテクチャが定義されている。しかしこれらのアーキテクチャは別々の要素として設計されており、認証認可システムを利用するためには、これらを組み合わせて認証認可システムを設計しなければならない。また認証認可システムの認証認可に関する運用は各組織の運用に順ずるとしていることから、従来の認証認可システムのアー

キテクチャをもとに構築されることになる。そのため組織のポリシーとシステムの管理運用を両立を実現することができない。

3.4. 従来の認証認可システムの設計プロセス

従来の認証認可システムのアーキテクチャによる、現在の認証認可システムの設計プロセスについて 3.1 節の 1、2 の場合とサービスが認証認可システムが一体となっている場合と独立している場合を例に示す。

まずサービスと認証認可システムが一体となっている場合について示す。この場合は Web サーバーに認証認可の機能を実装することになる。全体的な設計の流れとしては、以下の通りである。

1. Web サーバーの認証認可ライブラリの選定と導入と設定
2. アカウント情報を保存するデータベースの選定と導入
3. 認証認可ライブラリとデータベースの接続
4. 認証に用いるデータの選定とアカウントテーブルの設計
5. 認証要素の選定
6. 認可ポリシーの設計とアカウントテーブルへの反映
7. ログの設計とロギングの設定

それぞれのステップにおける詳細を述べる。1. 各 Web サーバーが有している認証認可用のライブラリからサービスのユースケースとアーキテクチャに合わせて選定を行い導入し各種設定を行う。この際に必要となる次にアカウントを保存し管理するためにデータベースを導入する。2. データベースの選定は数ある RDB の中からサービスのユースケースに合わせて、もしくは認証認可機能に合わせて選定を行う。3. 導入したサービスとデータベースの接続を行いデータ送受信のテストを行う。動作しない場合、1. に戻り設定を確認を行う。4. 認証に用いるために必要となるアカウントのデータの選定をし、データベースに対してアカウント用

のテーブルを作成する。5. サービスで利用する認証方式を選定する必要がある。ただし利用しているWebサーバーのライブラリで利用することが可能な認証要素の中から選択することになる。しかしWebサーバーの認証認可ライブラリはあくまでもWebサーバーの付属品であるため利用可能な認証要素は少なく、ワンタイムパスワードた多要素認証が実装されていないことが多く、専用のソフトウェアを導入する必要がある。6. サービスを利用するユーザーの権限をもとに認可に必要となる権限を設計し、4. で設計したアカウントテーブルに追加する。 7. 認証認可機能が利用された際にどのようなデータを記録するかを洗い出しログを設計する。またログのデータ形式や取得間隔などを設定する。このようにWebサーバーに認証認可機能を導入する場合、開発者自信が設計、設定しなければならない要素が多く、システムのアーキテクチャ設計、データベース設計やセキュリティ、システム監査の技能、ノウハウを有していなければならない。そのためアーキテクチャやセキュリティの設計不備が発生する可能性が高い。これらの技能、ノウハウを補うために設計を行う際に開発者は、Webサーバーの認証認可ライブラリのドキュメント [68] やエンジニア向け情報共有サービスにおいて該当するノウハウを取得し、それぞれの断片的な情報を組み合わせて設計を行う。

次にサービスと認証認可システムを分けて構築する場合について示す。この場合は認証認可システムを実装することになる。全体的にな設計の流れとしては、以下の通りである。

1. 認証認可で利用するプロトコルおよびソフトウェア (IdP) の選定、導入
2. アカウント情報を保存するデータベースの選定と導入
3. IdP とデータベースの接続
4. 認証要素の選定
5. 認可ポリシーの設計と IdP への反映
6. ロギングの設定

それぞれのステップにおける詳細を述べる。1. サービスと連携するための認証認可で利用するプロトコルを選定し、認証認可機能を有しているソフトウェア (IdP)

の選定を行い導入する。2.IdP で用いるアカウント情報を保存するデータベースの選定、導入を IdP のドキュメントをもとに行う。3.IdP のドキュメントをもとに IdP とデータベースの接続の設定を行う。IdP を活用する場合は接続時に自動的にデータベースに認証認可で必要となる情報のテーブルが作成される。4. サービスで利用する認証方式を IdP 上で選定する。5. サービスを利用するユーザーの権限をもとに必要となる権限を設計し、IdP 上で反映する。6.IdP が利用された際に出力するログのデータ形式や取得間隔などを設定する。サービスと認証認可システムを分けて構築する場合、認証認可機能が網羅的に実装されているソフトウェアである IdP を導入するため、認証認可に必要となる機能を有するライブラリの選定や導入、データベースの設計やセキュリティ、システム監査のためのログの設計といった技能やノウハウが必要となる作業を行う必要がない。しかし認証認可システム全体のアーキテクチャの設計については開発者のノウハウが必要となる。しかしこのような認証認可システムは数多く導入されている事例が公開されている。これにより認証認可システムのアーキテクチャや各種構成要素の選定をこれらの事例をもとに設計することが可能である。公開されている事例としては、学認の IdP 構築・運用手順書 [17]、NIST SP800-63-3 [10] が挙げられる。しかしこれらの事例は事例を公開している組織の環境に適したものとなっているため、必ずしも自組織に適している設計であるとは言えない。

このように認証認可システムの設計は様々な構成要素と利用するソフトウェア、システムのアーキテクチャなど考慮しなければならない点が多く複雑である。そのため実装事例をもとに設計を行うことが有効であるが、それぞれの構成要素の依存関係が強いため実装例で利用されている以外のソフトウェアを利用することは難しい。

3.5. 従来の認証認可システムの運用の課題

3.4 節で認証認可システムの設計プロセスについて述べた。次に認証認可システムを実際に構築し運用するための課題について述べる。

まずシステムの管理運用に必要となる要素は、システムのリソース状況とログ

による状態の把握、異常な状態の際の適切な対応である。さらに認証認可システムではユーザーのアクセス状況を記録するアクセスログによるアカウント利用状況の把握が必要となる。そのため運用者はこれらを実施するための技能とノウハウが必要となることが課題である。この課題を解決するために公開されているシステム運用事例やガイドラインを活用する方法がある。しかしこれらのガイドラインには、管理運用を行うためにどのようなデータを収集するべきか、どのような対応をすべきかという内容しか記載されておらず、実際にシステムに対してどのような操作をどの要素に対して行えばよいかについては言及していない。理由としてはシステムを構成しているソフトウェア毎に操作方法や実装されている機能が異なるためである。さらに認証認可システムのアカウントの認証と認可に対する管理運用については自組織のポリシーや環境、展開しているサービスによって大きく異なるため、ガイドラインは存在しているもののそのまま適用することは困難である。

3.6. 従来のアーキテクチャでの課題解決のための先行事例

3.6.1 依存関係が少ないシステムアーキテクチャの実現

複数の機能を持ったソフトウェアを有するシステムを構築する際の複雑な依存関係の課題を解決するための事例について述べる。一般的にマシンにソフトウェアをインストールする際には “apt“、“yum“、“rpm“などのパッケージ管理ツールを活用し直接インストールを行う。この時必要となる環境依存ソフトウェアや設定ファイルは自動的にマシンに作成される。この手法の問題点としてはマシンの環境が複雑になることが挙げれる。マシン内の様々な場所にフォルダやファイル、環境変数が作成され、依存関係にあるバージョンが異なる同一のソフトウェアやライブラリが増え、それにより予期せぬ問題が発生したり、デバッグが困難になるといった課題がある。

この課題を解決するために、サーバークライアントモデルを参考にし、ソフト

ウェアや機能ごとにシステムをマシン単位で分離しそれぞれのマシン間を API で連携する手法がある。これにより依存関係がソフトウェアごとに整理された環境を作ることが可能である。しかし課題としては利用する要素が増加するほど必要となるマシンが増加することである。この課題は単一マシン内で仮想的に複数のマシンを作ることが可能となる ESXi [69] のような VM 技術を用いることで解決することが可能である。また近年ではクラウド基盤サービスを活用することで物理的なマシンを用意しなくても同様の効果を得ることが可能である。しかし課題としては管理運用を行うためには利用しいているマシンにアクセスしなければならないため運用者の負担が増加する。これについては、3.6.2 節、3.6.4 節に示す。

この課題を解決するために、仮想的にマシンを作成するのではなくマシン内に複数の仮想的な OS を作成する手法が用いられている。代表的なものに Docker [70] や Podman [71] が挙げられる。これにより課題となっているソフトウェア毎での環境の分離を一つのマシン内で行うことができる。これにより依存関係についての課題は解決可能である。しかし課題として管理運用に専門的な知識とノウハウを要するため困難であることが挙げれる。この課題を解決するために Google が開発、提供している Kubernetes を代表とする管理運用基盤は存在しているものの、Google のデータセンターのような規模で利用することを想定しているため、この基盤を管理運用することが非常に困難であり、マネージドクラウドサービスを利用することが前提となっている。

これらから本研究の認証認可基盤では、管理運用の手法が定義されたソフトウェア毎に分離されているシステム構成を実現する。

3.6.2 認証認可システムのシステム状態把握

認証認可システムが現在どのような状態であるかを把握することについての現状を、状態を把握するために実装されている仕組みやプロトコル、また実際に認証認可システムを運用する場合の現状について示す。

まずシステムの状態とな何か、また状態を取得する原理について説明する。システムの状態とはプログラムを動作させるために用いる OS リソースの状態を意味しており、CPU、I/O ポート、割り込み番号、メモリー、ディスクなどのコン

ピュータを構成する要素の状態を表している。これらの状態を運用者が把握する方法として Linux 系 OS ディストリビューションでは、“ps(ps aux)“、コマンドを用いることでシステムコールからシステム状態を取得することが可能である。この時取得可能な情報は以下のとおりである。

- USER プロセスの所有ユーザー
- PID プロセス番号
- CPU CPU の占有率
- MEM 実メモリでの占有率
- SIZE 仮想分も含めた使用サイズ (K バイト)
- RSS 実メモリ上の使用サイズ (K バイト)
- TTY 端末名
- STAT プロセスの状態
- R 稼動中
- S 一時停止中
- D 停止不可能で一時停止
- T 終了処理中
- Z ゾンビプロセス
- W スワップアウト
- N nice 値
- START プロセスの開始時刻
- TIME プロセスの総実行時間

- COMMAND 実行コマンド名とパス

この手法により得られる情報はコマンドを入力した瞬間のシステムの状態となっており、継続的にモニタリングをし続けることを想定されていない。継続的にモニタリングするためには、“ps“コマンドを継続的に実行し続け、継続的に結果を表示し続けなければならない。例としては、引数に指定したコマンドを一定間隔で実行し続け結果を表示し続ける “watch“、‘top‘コマンドが挙げられる。監視例を図 3.1 に示す。この手法の課題を挙げると、システムの状態を把握するためにはシステムを導入しているマシンにアクセスする必要があることであり、監視対象のマシンが 100 台あれば 100 台すべてにアクセスしなければならない。また運用者がアクセスして該当コマンドを入力するまで状況を把握することができない。つまり運用者が状況を確認しようというアクションを起こさない限りは状況を把握することはできない。

図 3.1　watch ps コマンドにより監視例

コマンドによる状況の把握における対象となるマシンすべてにアクセスし、運用者がコマンドを入力しなければならないという運用の負担を解決するために、Zabbix [72] や Metricbeat [73] を代表とするセンサーの存在が挙げられる。導入例を図 3.2 に示す。センサーとはシステムの状況を把握したいマシンに事前にイン

ストールすることでシステムのバックグラウンドで状況把握のコマンドを実行し尚且つ、TCP を用いて外部に特定のデータ形式で送信する機能を有しているソフトウェアの総称である。このセンサーをインストールすることでマシンにアクセスせずに、データを一か所に収集することが可能になっている。この際のデータ構造と形式は利用するセンサーの実装によって異なっており、特定の標準は存在せず各社で自由な実装を行っている。収集方法についても特定の標準は存在せず利用するセンサーに依存する。Zabbix の場合は、Zabbix サーバーのデータベースコンポーネントにより収集され、Metricbeat の場合は、ログ収集ソフトウェアの“Logstash“ [74] によって収集される。このように各マシンのシステムの状況を一か所に収集することが実現されている。ただしこの場合においてもコマンドを活用した場合と同じように、運用者が状況を確認しようというアクションを起こさない限りは状況を把握することはできない。

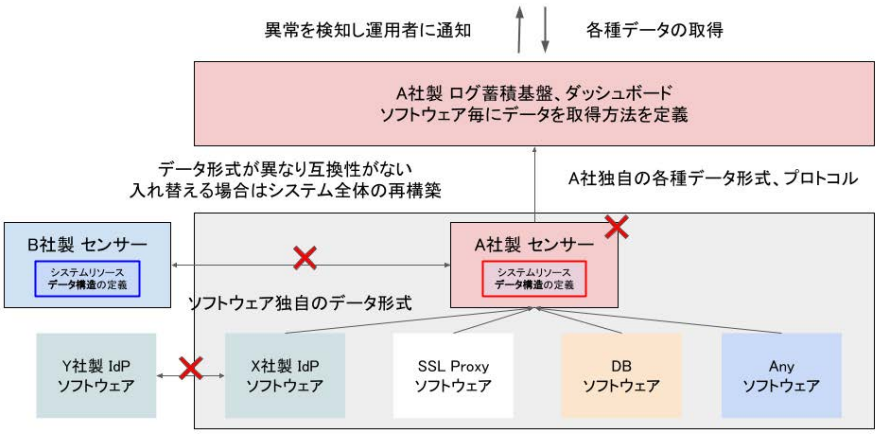

図 3.2　センサーを用いたデータの取得

運用者が状況を確認するアクションを起こさなくとも、システム側から運用者に状況を通知する方法についても示す。ここでいう運用者に通知するとは、email や slack などのコミュニケーションツールに対して異常な状態であることを知らせることを想定している。現状としてシステムに異常が発生した場合に運用者に対して状況を通知するためのプロトコルは定められておらず、システムの外に別途

通知するためのプログラムを実装する手法が用いられている。具体的には、センサーを活用して収集したデータに対して異常を検出するための閾値を設定したり脅威を分析する仕組みを導入することでシステムの状態の異常を検出し、コミュニケーションツールの API を活用して運用者に通知する手法である。実現例としては、AWS 上のシステムリソースに対してシステムの状態の可視化や監視、管理者への通知などの総合的な機能をもつ AWS CloudWatch [75] や、GCP 上における Cloud Monitoring [76] が挙げられる。しかし本手法の問題点としては、AWS や GCP のシステムを管理運用するために特化した仕組みとなっているため、他社のクラウドサービスや自組織においてオンプレミスに実装されているシステムに対しては互換性がなく利用することができない。

ここまでをもとに認証認可システムにおける適用について示す。まず 2.6 節で示した IDaaS の場合について述べる。IDaaS における認証認可システムの状況の把握については、AWS や GCP において実装されている状況を把握するための仕組みと同様に自らのサービスを構築しているシステムの状態を把握すること特化した仕組みとなっている。そのため例として Auth0 や Okta が実際に利用している状況把握手法を公開したとしても、本研究のターゲットである自組織で運用する認証認可システムに対して互換性があるものとはなっていない。次にセンサーを用いる場合について示す。結論としてはセンサーを認証認可システムに適応することは可能であり、センサーを用いることで認証認可システムのシステム稼働状況を収集し一元的に管理することが可能となっている。また認証認可システムを構成する構成要素や利用しているソフトウェアが異なる場合においても同様の効果を得ることが可能な汎用性を実現している。

ここまでで現状の認証認可システムにおけるシステム稼働状況の把握については、システム側から運用者に対してシステムが異常な状態であることを通知を汎用的に行うことは実現されていないが、システム稼働状況を表すデータを集約し運用者が状況を把握することが可能となっている。ただしデータの集約方法およびデータ形式についてはセンサー毎に独自の実装となっているため、センサーとデータを集約するシステムの互換性がなくベンダーロックインが発生する課題がある。

これらから本研究の認証認可基盤では、認証認可基盤の状態を把握するための汎用的なデータ構造と取得方法を実現する。

3.6.3 認証認可システムのアカウントの状態の把握

認証認可システムのアカウントの状態を把握するためのアクセスログとその活用方法について示す。認証認可システムにおけるアクセスログはユーザーがログインを行う際に作成されるログであり、アクセス日時、アクセス元 IP アドレス、ログインイベントの詳細、認証メッセージなどのメタデータの集合帯である。このログの主な活用方法としては認証および認可を行う際に異常が発生していないかの判断を行うことに必要不可欠である。アクセスログについての現状について述べる。

まずアクセスログに含まれているデータについて述べる。アクセスログに含まれているデータ種類およびデータ構造についての標準は存在していない。そのため認証認可システムに利用されているソフトウェアによって異なっている。例えば keycloak [77] では、Java EE アプリケーションサーバである JBoss [78] を利用しているため、アクセスログは RFC3164 で定義されている syslog のファシリティコードと重大度レベルを参考にしたデータ選定を行っている。データ構造については独自のフォーマット (正規表現:%dyyyy-MM-dd HH:mm:ss,SSS %-5p [%c] (%t) %s%e%n)、JSON、XML となっている。次に学認等で用いられている Shibboleth [17] では、アクセスログが標準で用意されておらず、自組織の認証認可のポリシーやアーキテクチャに合わせて運用者がアクセスログを設計し定義する必要がある [79]。また IDaaS である Okta では、アクセスログを LogEvent object [80] という独自の構造体で定義しており、Okta を利用することに特化したアクセスログを定義している。

次にアクセスログを活用するにあたり取得方法について現状を述べる。アクセスログはユーザーアクセスが発生する度に認証認可システムが構築されているマシンのログ保管ディレクトリに保存されており、運用者は対象マシンのログ保管ディレクトリにアクセスし標準入出力コマンドおよびエディタを活用することでアクセスログの詳細の確認を行う。このとき運用者が取得するログの例を図 3.3 に

示す。

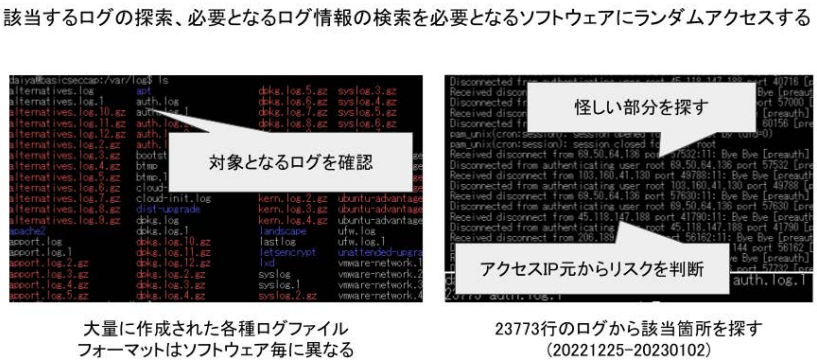

図 3.3　運用者が取得するログの例

これにより認証および認可を行う際に異常が発生していないかの判断を行うことに必要不可欠であるアクセスログを運用者が取得し確認することが実現できる。しかし図 3.3 で示している大量のデータから運用者が異常であるかどうかを判断することは困難である。そこでデータを運用者がアクセスログを活用する情報へ変換することが必要という課題を解決するために、アクセスログに含まれているデータを時系列毎のグラフや地図で可視化し運用者にダッシュボードとして提供する手法が用いられている。例として、Kibana [81] ではデータベースに保存されている json 形式のデータをパースしてダッシュボード上に展開、用意されているグラフ、地図、強調表示などの視覚表現機能に反映させることでダッシュボードを作成することが可能なソフトウェアが挙げられる。これによりデータの中身に関係なくダッシュボードが読み取ることが可能なデータ形式であれば運用者が活用可能なアクセスログの提供を実現することが可能である。

ここまでで認証認可システムにおけるアクセスログと活用方法については、認証認可システムへのアクセス毎にアクセスログを作成し保存することで、認証および認可を行う際に異常が発生していないかの判断を行うための判断材料となっている。またログ形式では運用者が理解することが困難であるという課題に対し

ては、特定形式のデータを展開、整形し視覚表現を追加するダッシュボードソフトウェアによって実現されている。しかし課題としては、運用者の状況判断までの流れは実現されているもののアクセスログが含むべきデータの種類やデータ構造については定義されていない。そのため利用する認証認可システムのソフトウェアに特化したアーキテクチャとなる。これにより利用している認証認可システムのサポート切れ、セキュリティポリシーや脆弱性の発生、新しい認証認可技術の登場といった場合による認証認可システムを構成しているソフトウェアの変更を行う場合、これまで利用していたアクセスログの活用に関するすべてのアーキテクチャを見直し再構成しなければならいという課題がある。

これらから本研究の認証認可基盤では、アーキテクチャや利用しているソフトウェアに依存しないアクセスログの取得と活用を実現する。

3.6.4 認証認可システムの管理運用の実施

認証認可システムに対して運用者が行うアクションは認証認可システムに問題がある場合と認証認可システムが管理しているアカウントに問題がある場合に行う。まず、認証認可システムに対して状況を改善するためのアクションをどのように起こすのかについて述べる。運用者は認証認可システムが構築されているマシンにアクセスする。次に問題の原因となっているソフトウェアに対してソフトウェア独自の操作方法に従って、問題を解決するために必要とあるコマンドを入力し解決を図る。例として証明書の期限が切れている場合には、SSL Proxy を停止したのちにマシン内の証明書の更新作業を行い、再度 SSL Proxy を起動するというアクションを起こす必要がある。つまりこのアクションは証明書の更新、SSL Proxy の操作方法のノウハウがある運用者でなければ行うことができない。次にアカウントに問題がある場合である。ここではアカウントを一時的に停止するアクションを例に挙げる。運用者は IdP の管理運用を行うインタフェースにアクセスをする。このインターフェースが GUI の場合はダッシュボードを操作することで、CLI の場合は必要となるコマンドを実行する。その後アカウントが停止されていることを実際にアクセスすることで確認を行う。ここでの課題としても先と同様に各インターフェースの操作方法を理解していなければアクションを起こすこ

とができないことである。

このようにアクションを起こすために複数の手順がある、またノウハウが必要となる場合にアクションを記録し再利用するという手法で解決することが可能である。これはノウハウを有している運用者がアクションの一連の流れと実行する操作をコードに置き換え記録しておくことで、ノウハウを有していない運用者であっても、このコードを実行すれば同じアクションと実施することができるというInfrastructure as Codeという手法である。古くはシェルスクリプトやバッチが挙げられ、近年では細かいファイル操作や結果の出力などの機能を有するAnsible [82]が挙げられる。しかしこの手法の課題としては、ノウハウがある運用者でなければコード化することができないことである。つまり組織内に最低でも一人はノウハウを有する運用者がいなければならない。また現在の運用のコード化は自由度があまりにも高いため、コード化するためのノウハウの取得の難易度が高く継続的に運用するハードルが高いという課題がある。

これらから本研究の認証認可基盤では、運用の際のアクションを基本的なシステム運用の知識を有している、つまり専門的な認証認可システムの知識を有していなくとも実施可能となるインターフェースを実現する。

3.7. プレーンとシステムアーキテクチャ

このような複雑なシステムの管理や運用のノウハウの問題を解決し、実践的に複数のシステムを管理するための先行事例としてネットワーク機器におけるプレーンと呼ばれる概念と機能、そのアーキテクチャについて述べる。

3.7.1 ネットワーク機器におけるプレーン

ネットワーク機器におけるプレーンとは特定の処理が行われる場所、機能を意味している。プレーンの種類としては、ネットワーク機器間におけるデータのやり取りを行う場所、機能をデータプレーン、やり取りされるデータをどこに転送するのかを制御する場所、機能を制御プレーン、そして操作方法や実装のことなる様々なベンダーのネットワーク機器を管理するために標準化された管理プレー

ンがある。これらによりネットワーク機器およびネットワークの実践的な管理運用を実現している。

次に、それぞれのプレーンの役割と各プレーン同士の連携について述べる。

3.7.2 データプレーンと制御プレーンの役割と連携

ネットワーク機器における、データプレーンと制御プレーンの役割および連携を図 3.4 に示す。まずベンダーの異なるネットワーク機器間において通信を実現するためにデータプレーンが定義されている。データプレーンでは実際にパケットの転送を行うことが役割となっている。しかしデータプレーンだけではどこの対象にパケット転送するのかの管理を行ってはおらず、このルーティングは制御プレーンで定義している。制御プレーンでは、どのようにどこにパケットを転送するのかを管理しており、さまざまなプロトコルを用いてネットワーク経路を特定し、これらの経路をルーティングテーブルとして FIB に格納してデータプレーンに渡すことで、ルーティングを制御している。

またこのアーキテクチャの利用例として、ソフトウェアでネットワークを定義し管理する SDN(Software Defined Networking) があり、世界中のネットワークや研究において利用されている [83]。

現在までのトラフィックの増加と SDN におけるベンチマークの研究 [84] [85] [86] からも示される通り、このようにルーティングとデータ転送を切り分け、複雑な処理を他の機能を有するプレーンに引き渡すことで、効率的なネットワークの管理を実現している。

3.7.3 制御プレーンと管理プレーンの役割と連携

3.7.2 項を採用し、相互運用性のあるネットワーク機器が様々なベンダーにおいて開発され利用されることで今日のネットワークを支えている。また利用するネットワーク機器が増加し多様なベンダーのネットワーク機器を管理ししている。しかしネットワーク機器を管理する方法については定義されておらず各ベンダーに

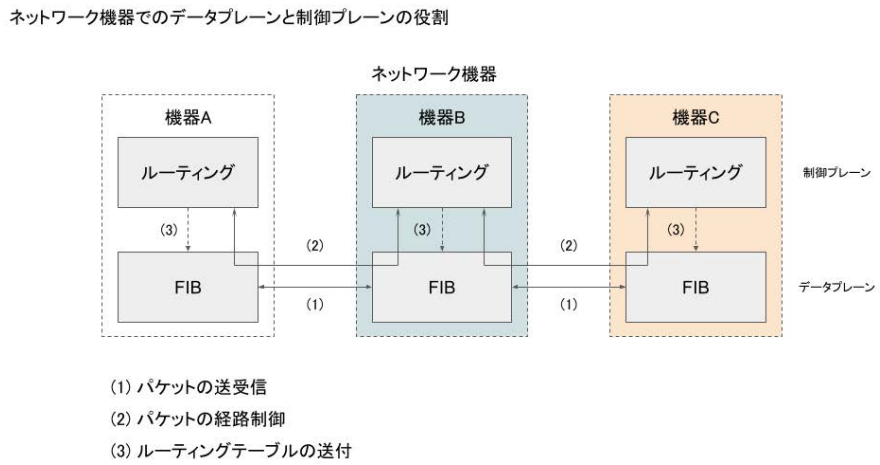

図 3.4　データプレーンと制御プレーンの機能と連携

よって設定やログの表現が異なっており、図 3.5 に示すようネットワーク機器自体を効率よく管理できないという課題が生まれた。

そこでネットワーク機器を管理することを目的として管理プレーンを新たに定義することで解決を図った。図 3.6 に示すように、ネットワーク機器の設定やログを MIB [87] という構造化された変数の集合で定義し、各機器から MIB を送受信するための SNMP を定義することで共通の方法によるネットワーク機器の監視を実現した。

また SDN では、管理プレーンとしてネットワーク機器を監視するだけではなく、制御プレーンの操作についても共通の操作方法である OpenFlow [88] [89] を定義することでネットワークの管理運用の共通化についても実現している。

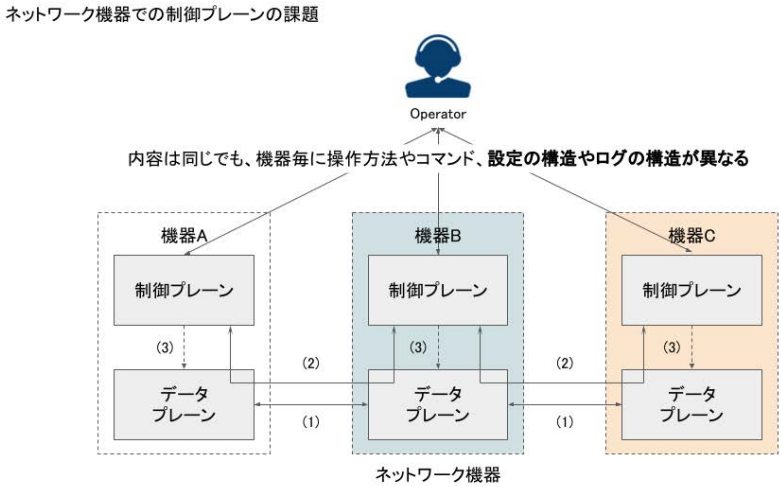

図 3.5　制御プレーンの課題

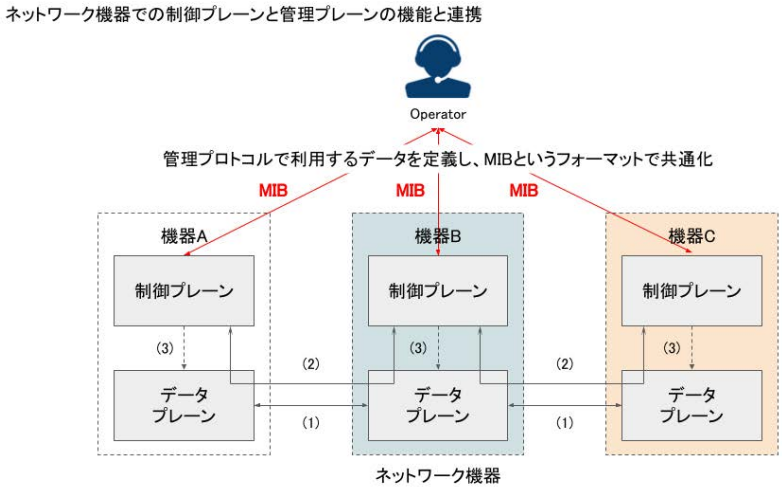

図 3.6　制御プレーンと管理プレーンの機能と連携

第 4 章
管理運用統合プレーンアーキテクチャの設計

3.2 節で述べたターゲットに対して課題を解決するための認証認可基盤および管理運用統合プレーンアーキテクチャの設計について述べる。本研究ではこれを実践的な認証認可基盤の実現のための管理運用統合プレーンアーキテクチャの実現と呼び、実現のための必要となる要素を整理を行う。

4.1. 認証認可基盤の実現のための管理運用統合プレーンアーキテクチャの提案

3 章で、現状の認証認可システムのアーキテクチャの課題を実際に広く一般的に運用されている認証認可システムを例に示した。またこの課題を従来の認証認可アーキテクチャの中で解決を図る先行事例を示し、従来の認証認可システムのアーキテクチャでは、ポリシーの反映とシステムの管理運用の双方を同時に実現することは困難であることを示した。

本研究では従来の認証認可システムのアーキテクチャを見直し、自組織のシステムポリシーを反映可能となる認証認可基盤、自組織の認証認可ポリシーの反映とシステムの管理運用をプレーンという概念により一元的に担う管理運用統合プレーンを定義することで、組織の認証認可ポリシーとシステムポリシーを守り、継続的に管理運用が行える、従来の認証認可システムのアーキテクチャとは本質的に異なる実践的な認証認可基盤の実現のための管理運用統合プレーンアーキテ

クチャを実現する。

4.2. 本提案のアーキテクチャの必要性

本研究の認証認可基盤および管理運用統合プレーンアーキテクチャの必要性について述べる。まず前提として組織の認証認可ポリシーとシステムポリシーを守り、自組織で運用可能となる認証認可システムが実現されるべきであると述べる。しかし 3.3 節、3.4 節の課題から示されるように認証認可システムの設計は専門的な知識を有している開発者でなければ行うことが困難であり、また実例やガイドラインをもとにした設計および運用は可能ではあるが、必ずしも自組織に適した認証認可システムであるとは言えない。理想的な認証認可システムとしては、自組織に合わせて自由にソフトウェアを選択して設計でき、どのような構成要素であったとしても適切に管理運用を行うことができるべきである。本研究ではこの自由に設計可能でかつどのような構成要素であっても管理運用が可能となる認証認可システムを認証認可基盤と呼ぶ。

次に管理運用統合プレーンアーキテクチャの必要性について述べる。現在、IDaaS や IdP を提供している企業や学認、Keio.jp などの認証認可基盤を有している組織では認証認可システムの管理運用が実現できている。しかし 3.5 節で述べた通り、認証認可システムは各組織の環境に依存した独自の管理運用を組織内の人材で行うことで実現している。また 3.3 節に示した通り、汎用的な管理運用が可能な認証認可基盤のアーキテクチャが活用されておらず管理運用を分野外としていることから、現状そのような汎用的にな認証認可基盤のアーキテクチャは存在していない。

これらのことからどのような組織においても汎用的に認証認可基盤と管理運用統合プレーンアーキテクチャを実現する必要があり、これが本研究の意義である。

4.3. 従来のアーキテクチャでの本提案の実現の可否

従来のアーキテクチャにおいて本研究のような認証認可基盤における管理運用統合プレーンの定義が行われていない要因としては、インターネット技術および業界がマルチステークホルダー化、細分化しそれぞれの専門性に特化した性能になっているため、他分野間の連携がうまく行われていないことが挙げられる。管理運用をサポートするためのサービスとして、Zabbix [72]、Prometheus [90]、Istio [91] を例にして詳細を述べる。

まずZabbixのコンセプトとしては、システムから取得可能なすべてのデータを集約し可視化するという点にある。これにより、システム運用に必要となるデータを統合的に取得数ることが可能となる。このデータ取得の圧倒的な自由度が売りの反面、どのデータをどのように扱うのか、また管理運用に必要となるデータは何であるのかという運用の本質部分に関しては考慮されていない。現在のデータ可視化ツールで見られるようなただ取得できるデータを収集し可視化しただけでありデータ自体には運用において何ら意味を持たないものとなっている。次にPrometheusについては、Zabbixの課題であるデータに対してオペレーションに利用するための意味付けをソフトウェア単位で行えるようになっている。Zabbixでは、すべてのシステムに対して同じセンサーを配置し、取得できるデータ取得するという方針で行われていたが、Prometheusではexporterと呼ばれるソフトウェアエンドポイント (センサー) を設置することでシステム毎に意味のあるデータを取得することが可能となっている。現在では、exporterは100種類以上公開されており、例えば、apache_exporterやmysql_exporter [92] が挙げられる。しかし、ソフトウェア毎に意味のあるデータを取得できるようになったにも関わらず、ダッシュボードにおける管理者へのデータの可視化はZabbixと同様のデータの単純な可視化となっている。最後にIstioについては、システムのモジュラーが進み始めた現在において管理すべきソフトウェア膨大になり、それぞれのデータ通信を行う依存関係がメッシュ状に複雑化し始めている。そのメッシュをサービスメッシュという。サービスメッシュの問題点としては、どこか一つのモジュール化したソフトウェアのトラフィックが詰まってしまった場合、システム全体の反応速度がその分遅くなってしまうことである。そのため、各システムのトラフィック情

報をリアルタイムに関しすることを目的として開発されたのがIstioである。Istioはモジュールの外側にプロキシを作成しデータを通過させることでトラフィックを監視することを可能としている。Istioの問題としては、あくまでもサービスのネットワークボトルネックを検出、検知するための仕組みであり、データプレーンの拡張として利用されるものとなっている。

このようにこれらのサービスはシステムの管理運用における特定の一部に特化しており、これを実際の認証認可基盤の管理運用に適用しようとした場合、管理者の能力や経験に大きく依存してしまう。また同様に、IETF Automated network management WG [93]、Cloud Native Computing Foundation [94] に挙げられるような、システムの管理運用の自動化や一般化を目指す世界的に巨大なカンファレンスや組織であっても、既存のネットワーク管理コマンドや仕組みの再利用が前提であったり、クラウドを中心とした管理運用が前提であるなど、それぞれの方針や管理運用を行うための前提が細分化され、専門的な領域に特化した管理運用を議論しており、実践に求められる複雑な環境化における管理運用統合プレーンは実現されていない。

4.4. 本提案アーキテクチャの実現の要件

実践的な認証認可基盤を実現のための要件について述べる。4.2節で示した通り、本研究における実践的な認証認可基盤のアーキテクチャは、認証認可基盤と管理運用統合プレーンで構成されている。それぞれの要件について次に述べる。

4.4.1 認証認可基盤の要件

認証認可基盤の要件として、以下の3つ挙げる。

- 自由なソフトウェアの選択が可能なアークテクチャ
- 構築環境への依存が少ないアーキテクチャ
- ソフトウェアに依存せずに管理情報を取得可能なアーキテクチャ

これらを満たすことにより、組織のシステムポリシーを適用可能でどのようなソフトウェアにおいても各構成要素の状況の把握可能となる汎用的な認証認可基盤を実現する。それぞれの挙げた要素について詳細を述べる。

まず自由なソフトウェアの選択が可能なアークテクチャについては、3.4 節で課題として挙げた通り、従来の認証認可システムは組織に合わせた自由なソフトウェアや構成要素を選択することができず、組織のポリシーを守ることができない。そこで認証認可基盤の構成要素をモジュールとして定義し、例としては IdP モジュールでは認証認可機能を提供するといった、構成要素をソフトウェアではなく機能として定義することで利用するソフトウェアの依存関係を解消する。次に構築環境への依存が少ないアーキテクチャについては、組織の構築環境や OS によって本研究で提案する認証認可基盤を構築することができないということは避けるべきである。そのために構築環境や OS に依存することなく、また構築手順が複雑とならない汎用的な認証認可基盤を実現する。最後にソフトウェアに依存せずに管理情報を取得可能なアーキテクチャについては、3.5 節で挙げた課題の通り、どのようなソフトウェアを選択してたとしても、ポリシーの反映とシステムの管理運用の双方を管理運用統合プレーンにより一元的に行うことが可能となるアーキテクチャを実現する。

4.4.2 管理運用統合プレーンの要件

管理運用統合プレーンの要件として、以下の 3 つ挙げる。

- 認証認可基盤からの管理情報によるシステムの状況把握
- 認証認可基盤からのアクセスログによるアカウントの状況把握
- 認証認可基盤に対する認証、認可、システムの運用の実施とポリシーの適用

これらを満たすことにより、認証認可基盤の継続的な管理運用を行うための管理運用統合プレーンを実現する。それぞれの挙げた要素について詳細を述べる。

まず認証認可基盤からの管理情報によるシステムの状況把握については、認証認可基盤の各構成要素の状況を一元的に管理することで、認証認可基盤全体を把

握することを可能にし、運用者にとって実践的な認証認可基盤のシステム管理を実現する。次に認証認可基盤からのアクセスログによるアカウントの状況把握については、認証認可基盤の IDP(Identify Provider) モジュールで管理されているアカウントの状態を一元的に管理することで、運用者が認証認可ポリシーとの比較を実践的に行うことを実現する。最後に認証認可基盤に対する認証、認可、システムの運用の実施とポリシーの適用については、システムの状況の把握、アカウントの状態の把握からシステムの運用と認証認可ポリシーを認証認可基盤に対して一元的に適応することが可能となるインターフェースを実現する。

4.5. 管理運用統合プレーンの役割と連携

認証認可基盤のプレーンアーキテクチャと認証認可基盤と管理運用統合プレーンの役割と連携について述べる。

4.5.1 データプレーンと制御プレーンの役割と連携

まず現状の認証認可基盤の構成要素をデータプレーンと制御プレーンとして当てはめ、それぞれの役割と連携について図 4.1 に示し整理する。

認証認可基盤におけるデータプレーンの役割は、各システム間で認証情報および Digital identity のデータを送受信することである。ここまでは、図 3.4 で示したネットワーク機器でのデータプレーン役割と似た働きをしている。

制御プレーンの役割については図 4.1 の (2)(3)(4) に示している通り、ネットワーク機器で定義されている制御プレーンとは異なり、それぞれ異なる役割を担っている。データプレーンで取り扱われるデータに対して、SSL Proxy ではデータの暗号化を SSL を用いて行い、IdP ではデータ取り扱う認証プロトコルの指定および制御、送信先サービスの制御、アカウント情報の制御を行う。DB においては SSRP を用いて、データを取り出す、保存するという制御を行っている。また制御プレーン同士の連携については、SSL Proxy-IdP 間では HTTP、IdP-DB 間では TDS を用いて連携をしている。

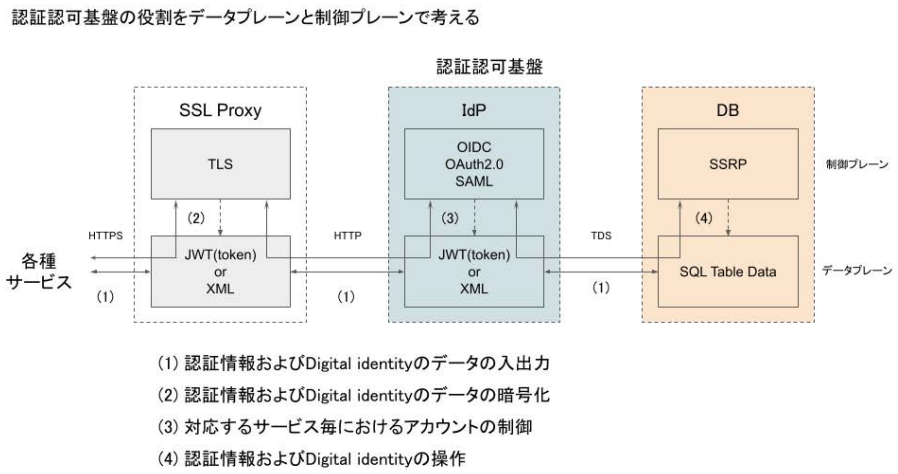

図 4.1　認証認可基盤でのデータプレーンと制御プレーンとしての役割と連携

4.5.2　制御プレーンと管理運用統合プレーンの役割と連携

このように認証認可基盤では、図 3.4 で示したネットワーク機器同士の連携とは異なり役割が異なる制御プレーンで構成されている。

次に認証認可基盤における制御プレーンの課題について示す。まず 3.7.3 項と同様にそれぞれの制御プレーンを管理する方法については定義されておらず設定やログの表現が異なっていることである。この課題は 3.2 節で示した本研究における解決するべき課題と一致し、ネットワーク機器における管理プレーンの役割と同様に管理運用統合プレーンを用いることで解決できると考えられる。しかし認証認可基盤では、ネットワーク機器のように同様の役割を持っているネットワーク機器を管理するという状況とは異なり、図 4.1 でも示した通りそもそもの役割が異なる制御プレーンで構成されている。そのため図 4.2 で示す通り、各ソフトウェアや制御プレーンで MIB のような統一したデータ形式やプロトコルを定義することは困難である。そのためネットワーク機器とは異なる設計の管理運用統合プレーンが必要となってくる。

そこで本研究では、この認証認可基盤における制御プレーンに必要となるデータや操作を整理し、図 4.3 に示すように認証認可基盤全体を効率よく管理するた

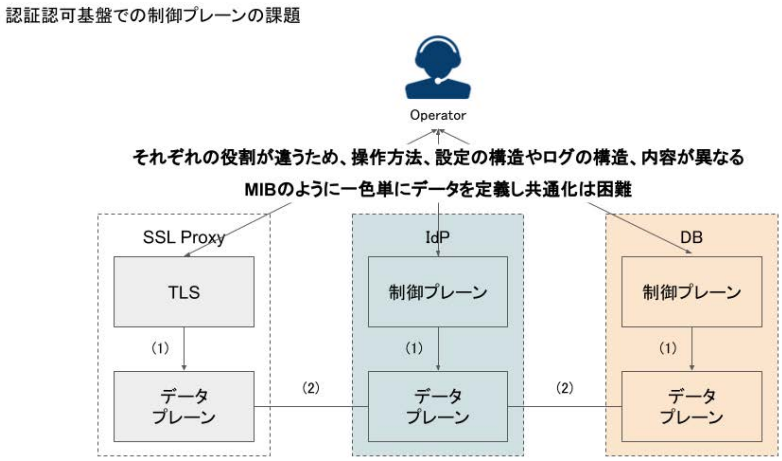

図 4.2 認証認可基盤での制御プレーンの課題

めの管理運用統合プレーンを設計し実装する。

4.6. 本提案アーキテクチャの設計とデータフロー

本提案アーキテクチャの認証認可基盤および管理運用統合プレーンの設計とデータフローについて示す。

4.6.1 本提案アーキテクチャの概要

本提案アーキテクチャの認証認可基盤および管理運用統合プレーンで必要となる機能について図 4.4 に示す。

まず管理運用統合プレーンは認証認可基盤の状況を一元的に把握する機能と認証認可基盤に運用を実施するためのインターフェースを有している。本研究では前者をログ基盤、後者を管理運用基盤と呼ぶ。次に認証認可基盤では認証認可基盤を構成している各要素のリソースデータ、ログの生成と構成する各要素の操作、認証認可機能およびアカウントの操作を行える機能を有している。またこれらの機能は各構成要素においてどのようなソフトウェアを用いても同様に機能する。こ

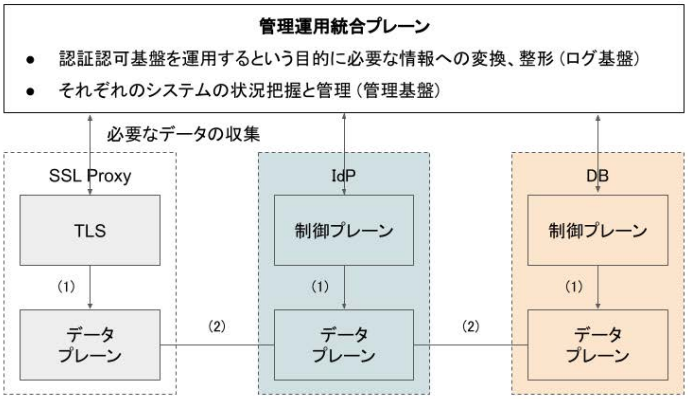

図 4.3 認証認可基盤での管理運用統合プレーンの定義

れらの機能は、ログ基盤と認証認可基盤を構成している各要素のリソースデータ、ログの生成が対に、管理運用基盤と認証認可基盤を構成している各要素の操作、認証認可機能およびアカウントの操作が対になっている。これらの機能を 3.6.2 項、3.6.3 項、3.6.4 項の課題を解決することで実現を図る。具体的な解決のための工夫および設計を次から述べる。

4.6.2 本提案アーキテクチャの設計と機能

管理運用統合プレーンのログ基盤の設計と認証認可基盤を構成している各要素のリソースデータ、ログの生成について述べる。

まずログ基盤では認証認可基盤の状況を一元的に把握する機能を実現する。この機能を実現するためには、認証認可基盤からリソースデータ、ログを取得、保存し運用者へ提供が必要となる。まずログの取得については、認証認可基盤にポーリングする方法と認証認可基盤から逐次データをパイプライン処理で受け付ける方法が考えられるが、一定間隔で取得するポーリングの場合、刻一刻と変化するリソース状態を把握することに適していない。そのため認証認可基盤でデータが生成されるたびにログ基盤に入力するパイプライン処理を行う設計とした。次に

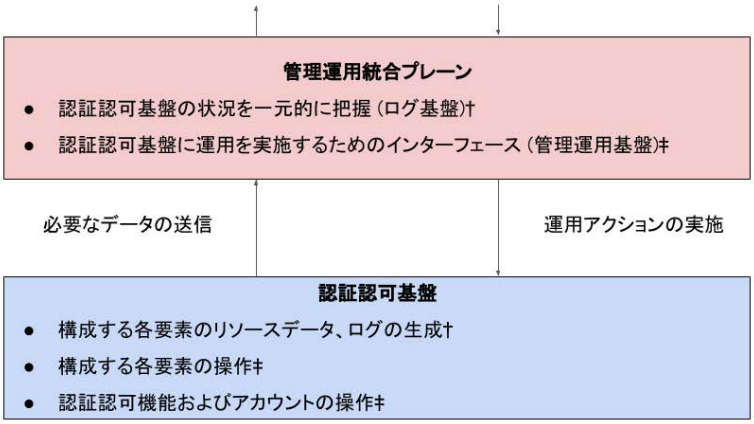

図 4.4 認証認可基盤および管理運用統合プレーンで必要となる機能

認証認可基盤を構成している各要素のリソースデータ、ログの生成については、3.6.2 項、3.6.3 項で示した通りソフトウェア毎に出力されるデータ形式が異なるため、従来の方式では自由なソフトウェアの選択が行えていない。そこで本研究ではシステムリソースやログを定義化し各構成要素から常に同じデータ形式の出力を得られるようにする工夫を行うことで、どのようなソフトウェアでも利用可能な認証認可基盤を実現する。具体的な設計としては、図 4.5 に示すように構成要素をモジュール化する。モジュールには各構成要素で選択されたソフトウェアと定義化されたデータ形式にシステムリソースやログを変換する機能を含ませ、これによりどのようなソフトウェアを選択しても常に全てのモジュールから定義化された同様のデータを得ることが可能となる。

次に管理運用基盤では認証認可基盤に運用を実施するためのインターフェースを実現する。この機能を実現するためには、認証認可基盤を構成している各要素の操作、認証認可機能およびアカウントの操作が必要となる。まず認証認可基盤で必要となる操作を 3.6.4 で挙げたように IaC を用いて簡略化する方法を用いる。しかし IaC の問題点としては簡略化する操作を行うことができる運用者でなければ簡略化することができない課題があり、従来の方式では本研究のターゲットが認証認可システムを管理運用する能力やノウハウがない運用者であるため簡略化す

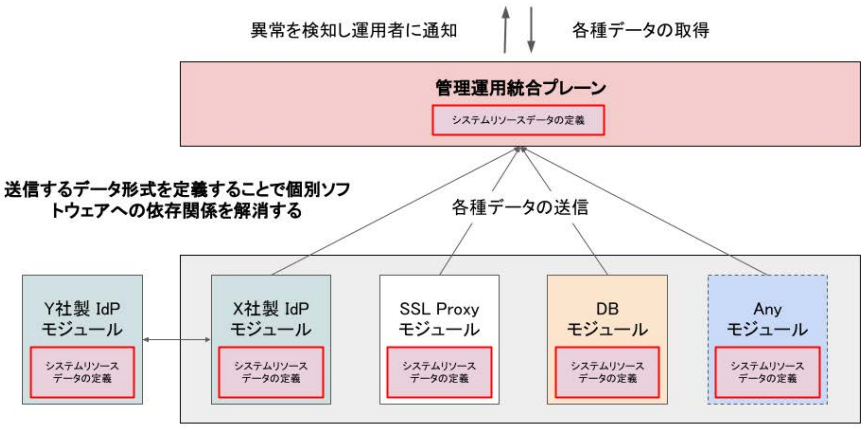

図 4.5 構成要素のモジュール化による各種データの定義化

ることができない。そこで本研究では、認証認可基盤で行う基本的な操作をユースケースとして洗い出し、事前に簡略化した操作をテンプレートとして管理運用統合プレーンの機能として持たせる工夫をすることで、管理運用統合プレーン上からテンプレートを実行することで行いたい操作を実行可能となるインターフェースを実現する。具体的な設計としては、図 4.6 に示すように管理運用統合プレーン上に認証認可基盤を運用するためのテンプレートの集合を管理する機能、テンプレートに対して環境変数を適用可能とする工夫をすることで、どのようなモジュールに対しても認証認可基盤を運用するアクションを専門家でなくとも実行することが可能となる。

このように、認証認可基盤としてはの各要素のモジュール化、状況を把握するデータの定義化、管理運用統合プレーンとしては各モジュールからのデータを提供するインターフェース、認証認可基盤運用テンプレートの活用を行うことが可能となる設計を行った。これにより、各組織に合わせた認証認可基盤のソフトウェアの利用と汎用手な認証認可基盤の運用を実現する。

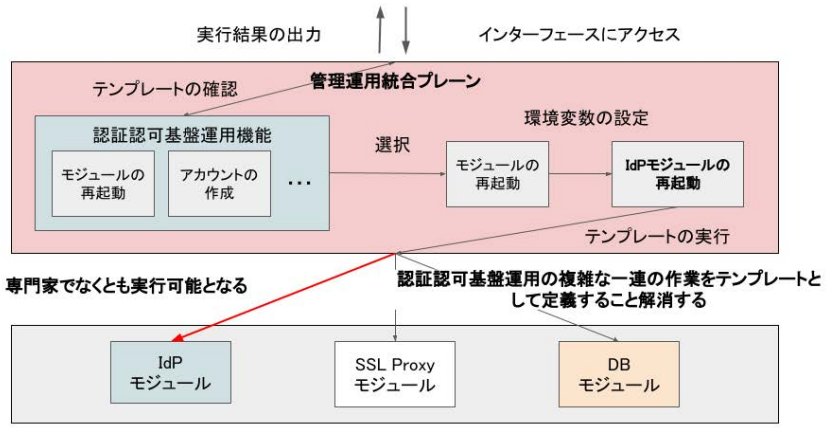

図 4.6 テンプレートを実行するインターフェースの実現

4.6.3 本提案アーキテクチャの連携とデータフロー

次に 4.6.2 項の設計をもとに本提案アーキテクチャの認証認可基盤と管理運用統合プレーンとの連携について示す。

まず管理運用統合プレーンから認証認可基盤に向けての連携とデータフローについて図 4.7 に示す。管理運用統合プレーンから認証認可基盤へ送信するデータは管理運用統合プレーンで定義されているテンプレートである。テンプレートは yaml 形式のファイルで構成されており、このファイルを認証認可基盤に送信する必要がある。そのためファイル送信プロトコルとして SFTP を活用して連携を行う。この際に管理運用統合プレーンでは事前に連携するマシンの SSH の接続情報を設定する必要がある。これにより管理運用統合プレーンから認証認可基盤へのファイル送信を実現する。データフローとしては SFTP でテンプレートを対象マシンに送信したのちに、対象モジュールに対して再び SFTP でテンプレートを送信する。その後、モジュール内でテンプレートが自動実行され利用したテンプレートが削除されるという流れとなる。この連携とデータフローについては、すでに標準化されているプロトコルのみを活用することでどのようなモジュール内のソフトウェアに対しても汎用的に活用可能な工夫を行っている。

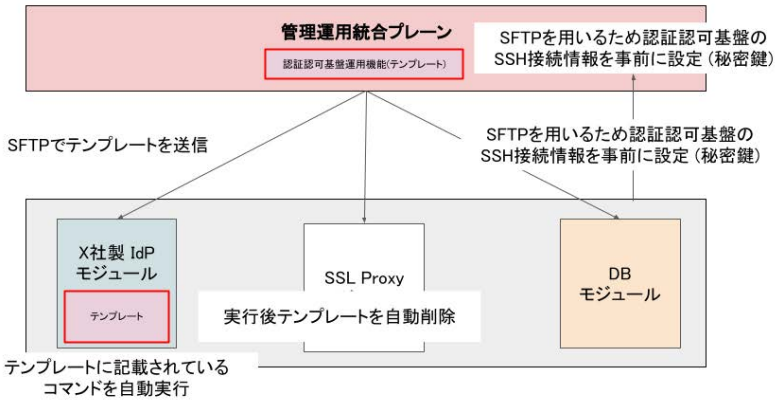

図 4.7 管理運用統合プレーンから認証認可基盤に向けての連携とデータフロー

次に認証認可基盤から管理運用統合プレーンに向けての連携とデータフローについて図 4.8 に示す。認証認可基盤から管理運用統合プレーンへ送信するデータは認証認可内のモジュールで定義されているシステムリソースデータおよびアクセスログである。この際の連携方法としては、ログをネットワーク上で送信するためのプロトコルである syslog を用いることで実現する。そのため認証認可基盤から管理運用統合プレーンへ送られるデータ形式はログファイルでなければならない。そこでモジュール内で出力されるリソースデータを単位時間当たりにログファイルに整形する工夫を行うことで、モジュールから管理運用統合プレーンにリソースデータを送信することを実現している。管理運用統合プレーンに届いた各種ログデータは 3.6.2 項で挙げたログファイルの探索の困難性を解決するために、定義されているシステムリソースデータおよびアクセスログでフィルターをかけることで選別を行う。これにより運用者は管理運用統合プレーンから必要となるデータを用意に取得可能となる設計の工夫を行っている。

4.6.2 項と 4.6.3 項により、本研究における管理運用統合プレーンを導入した認証認可基盤の設計とする。

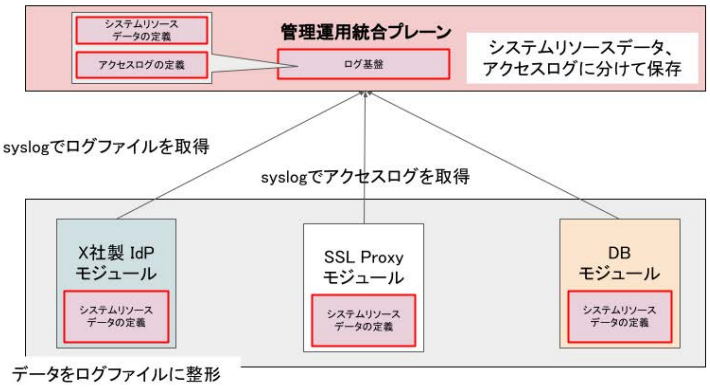

図 4.8　認証認可基盤から管理運用統合プレーンに向けての連携とデータフロー

4.7. 本提案アーキテクチャのシステム構成

4.6 節の設計をもとに、本研究において設計した認証認可基盤および管理運用統合プレーンアーキテクチャと認証認可基盤および管理運用統合プレーンの構成要素におけるデータフローについて示す。まず、本研究の課題としている認証認可機能を持つシステムのアーキテクチャについて示し、次に本研究において設計した認証認可基盤と管理運用統合プレーンについてのアーキテクチャを示す。また各システムおよび機能間でのデータフローや利用しているプロトコルを示す。

まず課題となるシステムの全体のアーキテクチャを図 4.9 を例として示す。本アーキテクチャは、認証認可機能を内包したシステムのアークきてクチャとなっており、構築している Web コンテンツおよび認証認可機能を一つのシステムとして提供している。次に各ソフトウェアおよび機能間におけるデータフローについて示す。

(1) サービスを利用するユーザーとサービス間で Web コンテンツおよび認証認可にまつわる全データのやり取りが、HTTPS を用いて行われる。ここではプロキシを利用する例を示しているが、Web サーバーソフトウェア上に証明書を設置する場合もあるがデータフローおよび利用しているプロトコルは同様である。(2)

(1) で送信されたデータを内部ネットワーク上の認証認可機能に認証情報がHTTPを用いて送信される。(3) (1)(2) でユーザーから送信された認証情報の検索、Webコンテンツで用いられているユーザー毎のデータが送受信される。ここではTCPをベースにしたデータベース毎の独自のプロトコルを用いている。(4) ここではサービスにアクセスしてきたユーザーのアクセスログがプロキシのソフトウェア毎で定義されている形式でログが保存される場所 (/var/log) に出力される。またソフトウェア自体の動作やエラーなどをシステムログとして保存する。(5) では、Web コンテンツで利用しているソフトウェアや使用によって大きくことなるが、一般的には (4) と同様にアクセスログとシステムログを出力する。認証部分に関しては認証結果をアクセスログとして出力することが多い。(6) データベースについても利用しているソフトウェア毎に出力されるログが異なるが、システムの稼働状況、データベースクエリの詳細、利用されたSQL文や問い合わせの詳細が記録されている。ログの形式についてもデータベース毎に異なる。ここからは運用者が管理運用する場合のデータフローを示す。(7) では、運用者がサービスが動作しているシステムに管理者としてアクセスするインターフェースである。一般的には、SSH を用いてシステムにアクセスする。ここから次に挙げていくフローをシステム上で実行していくことになる。またデータをローカル環境に取得したい場合には、SFTP で各種ファイルをダウンロードすることも可能である。(8) 各種ログを確認するためのインターフェースであり、ログが保存されているフォルダにアクセスし必要となる各種ファイルにアクセスする。各種ファイルへのアクセス方法は標準出力やエディタを活用することとなる。また検索などについてもコマンドの組み合わせや正規表現を活用して全検索することになる。(9)(10) は、ソフトウェアが動作している設定ファイルへのアクセス、各ソフトウェアの起動、停止、再起動などのシステム操作を行う場合に利用するインターフェースである。この時の各種操作についてはソフトウェア毎に定義されおり異なる。(11) はデータベースへのアクセスである。保存されているデータ、テーブルの確認、新たなテーブルやインデックスの作成、削除、修正などのデータベース操作を行う。これについてもデータベースの種類によって操作方法が異なる。

次に本研究で設計したシステムの全体のアーキテクチャを図 4.10 を例として示

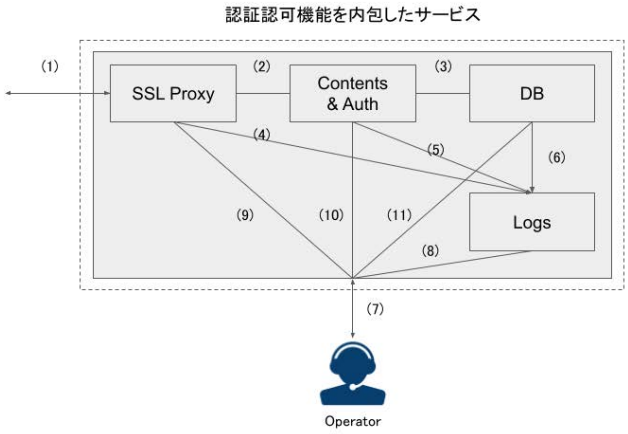

図 4.9　一般的なシステム全体のアーキテクチャ

す。本アーキテクチャは、認証認可機能の提供および個人情報を管理運用を行う認証認可基盤、各種構成要素からのシステムログおよびアクセスログを管理するログ基盤、認証認可基盤のシステムとしての管理運用、認証認可機能の運用、ログ基盤で収集されたログの検索可視化を行う管理運用統合プレーンで構成されている。この 3 つのシステムはそれぞれが独立しており、API を通して (4、5、6、10、13、14) 連携している。詳細な構成については認証認可基盤については 5 章に示す。次に各ソフトウェアおよび機能間におけるデータフローについて示す。

(1) 本アーキテクチャはサービスと認証認可基盤が独立している構成となっているため、ここではサービスからの認証情報のみが送信されてくる。認証情報は SSL でエンドツーエンドで暗号化されている。(2) (1) で送信されたデータを内部ネットワーク上の認証認可機能に認証情報が HTTP を用いて送信される。(3) (1)(2) でユーザーから送信された認証情報を検索しユーザー毎のデータが送受信される。ここでは TCP をベースにしたデータベース毎の独自のプロトコルを用いている。(4) ここではサービスにアクセスしてきたユーザーのアクセスログがプロキシのソフトウェア毎で定義されている JSON 形式で、ソフトウェア自体の動作やエラーなどをシステムログとして JSON 形式でロギングサーバーに送信する。また CPU、メモリ使用率、ネットワーク I/O などの現状のシステムの動作状況を

JSON 形式で送信する。(5) では、IdP へのアクセスログとシステムログを JSON 形式でログ基盤へ出力する。また CPU、メモリ使用率、ネットワーク I/O などの現状のシステムの動作状況を JSON 形式で送信する。(6) データベースについては利用しているソフトウェア毎に出力されるログが異なるが、システムの稼働状況、データベースクエリの詳細、利用された SQL 文や問い合わせの詳細が記録されている。ログの形式については JSON 形式である。また CPU、メモリ使用率、ネットワーク I/O などの現状のシステムの動作状況を JSON 形式で送信する。(7) ロギングサーバーで受信したログを保存し、かつ (8) で呼び出す際のログの送受信を行う。ログは (8) で検索するために JSON 形式で送受信される。(8) (9) で可視化される各種ログを JSON 形式で送信する。また (9) で入力され、WebAPI で送られてきた自然言語をもとにデータベースから JSON ベースで検索を行う。(9) WebAPI を利用して各種ログを JSON 形式で送信し、ダッシュボード上ではユーザーの定義したデータ展開方法にしたがって JSON をパースし可視化を行う。本研究では認証認可基盤に合わせたデータ展開スキーマを定義して実装しているためこちらを適応している。(10) 運用者に可視化したログをダッシュボードとしてウェブブラウザ上に HTTP/HTTPS で表示する。自然言語によるログの検索については WebAPI 経由でサーチエンジンに送信され、結果が表示される。(11) では、運用者が管理運用統合プレーンを操作するフローである。管理運用統合プレーンはダッシュボードとしてウェブブラウザ上に HTTP/HTTPS で表示する。リポジトリに保存されている管理運用のテンプレートコードを (12) で呼び出し、SFTP を用いて認証認可基盤、ログ基盤に対して送信して実行する。実行結果は標準出力としてテキスト形式で受信する。(12) (11) で利用するテンプレートコードを Git を活用して都度呼び出す。コードは HTTPS か SSH を用いて受信する。(13) (11) で選択されたテンプレートを SFTP で送信する。コードの実行は SSH を用いて Python でリモート実行される。(14) (11) で選択されたテンプレートを SFTP で送信する。コードの実行は SSH を用いて Python でリモート実行される。

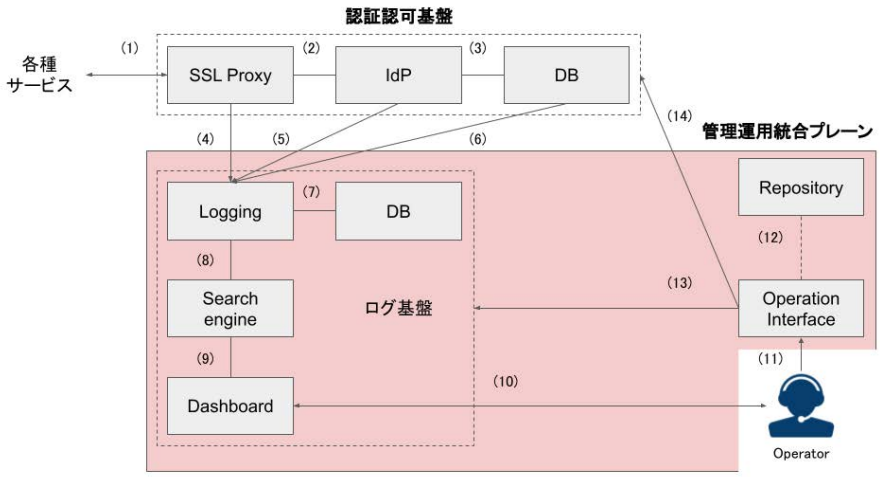

図 4.10　本研究で構築したシステム全体のアーキテクチャ

4.8. 本提案アーキテクチャで取り扱う管理情報

次に 4.7 節に示した図 4.10 データフローから本研究における認証認可基盤を管理運用していくために管理するべきデータを整理し選定する。本研究では管理するべきデータをシステム状況を示すデータとユーザーのアクセスログの 2 種類に分類する。

まずシステム状況を示すデータについて整理する。これは認証認可基盤を構成しているそれぞれのシステムが正常に稼働しているのか否かを判断するためのデータである。このデータ利用することで管理運用統合プレーンからシステムの起動、停止、再起動などの操作や利用しているコンテナスペックの調整等の運用を行うため指針となる。本研究においてはこれらの情報を Docker から取得し、システムの状態を把握するための以下の情報を利用することとした。設計したデータ構造および出力されるデータを付録 C に、実際に取得するデータの詳細については、5.2.2 項に示す。

- システムエラーログ (error.log)
- CPU、メモリ稼働率

- ネットワーク I/O(利用帯域)
- Disk 情報 (データ使用量、ディスク I/O)
- システムアップタイム

まず、システムエラーログ、Disk 状況については、システムが正常動作していないもしくはアラートを出している際の状況判断に用いる。CPU、メモリ稼働率、ネットワーク I/O については、システムの動作が鈍い状態やシステム自体は動作しているがアクセスができない等のサービス停止状態の状況判断に利用する。システムアップタイムついてはシステムを起動した際に何かしらのエラーによって再起動を繰り返していないか、正常に起動しているのかを判断するために用いる。

次にユーザーのアクセスログについて述べる。これは認証認可基盤をサービスから利用してくるユーザーのログイン状況やアクセス時間、アクセス回数などユーザーの行動を把握することでユーザーのアカウントに対する攻撃を把握することを目的としている。設計したデータ構造、ログ生成の設定ファイル、出力されるアクセスログを付録 D に示す。管理運用に必要となる情報は以下の通り。

- ログイン判定 (成功、失敗、アップデート)
- アクセス元 IP アドレス
- ログイン元サービス
- ログインに利用したユーザー名
- ログイン時間
- ログイン施行回数

それぞれの情報の役割としては、ログイン判定、ログインに利用したユーザー名によって、どのアカウントがログインに使用され成功しているのかを判断する。アクセス元 IP アドレスは、GeoIP を利用することで地理情報と紐づけどの国のどの地域からアクセスしてきているのかを取得することでユーザーが明らかにおかしい場所からログインされいないのかを判断する。ログイン試行回数は同一 IP ア

ドレスからのログ数を計測し、ログイン時間と照らし合わせることで単位時間当たりのログイン回数を取得する。これらの情報を利用することで最終的にアカウントが攻撃されているのかを判断することが可能となる。また、これらの情報とログイン元サービスを確認することでアカウントが狙われている原因を推測することに活用できる。

なおこれらの情報は、図 4.10 の “IdP“のアクセスログから取得する。詳細は、5.2.2 項で述べる。

4.9. 管理運用統合プレーンによる管理運用

図 4.10 で示した管理運用統合プレーンの各構成要素に対する役割について例を踏まえて示す。

4.9.1 認証認可基盤に対する管理運用

認証認可基盤に対する管理運用統合プレーンの役割としては、認証認可基盤を構成している各モジュールへのシステム操作、操作また IdP に保存されているアカウント情報の操作である。この役割に対する各モジュールはインターフェースとして Docker コンテナのコマンド操作、管理運用統合プレーンからの SSH でのアクセスを有している。まず認証認可基盤は図 4.10 で示す通り、“SSL Proxy“、“IdP“、“DB“で構成されおり、それぞれが Docker コンテナとして動作してる。次に管理運用統合プレーンにはこの各構成要素に対して状況の確認、操作を行うためのテンプレートが yaml で定義されており、実行することで状況の確認および操作を各構成要素に対して行うことができる (図 4.10 の 13)。例として認証認可基盤内の “IdP“を再起動する際の流れを図 4.11 に示す。まずモジュール内で実行したい操作を yaml で表現し登録を行う。この際のコードは管理運用統合プレーンに直接登録する方法と図 4.10 の 14 に示すリポジトリに登録しているものを利用する方法を選択することが可能となっている。実際に登録している再起動にまつわるコードの一部を以下に示す。

```
tasks/main.yml
  - name: Stop a container
  community.docker.docker_container:
    name: "{{ container_name }}"
    state: stopped

  - name: Start a container
  community.docker.docker_container:
    name: "{{ container_name }}"
    state: started
```

このコードはIdPコンテナを停止し、起動するという操作を意味している。次にこのコードをAnsibleを利用することで対象のモジュール内で実行する。ますSFTPを用いて対象となる“IdP“が動作しているマシンに対して送信し、続いてSSHを用いて同様にアクセスしたのちに、SFTPで送られてきたコードを用いて解釈することでコンテナを停止および実行を行う。実行後、実行結果が“IdP“から管理運用統合プレーンに対してSSH上で返却されることで実行した一連の流れを終了とする。

次に“IdP“に対して新規アカウントを作成する作業についての例について図4.14に示す。まずコンテナの再起動と同様に、モジュール内で実行したい操作をyamlで表現し登録を行う。アカウントの新規作成にまつわるコードについてはコード量が多いため付録Fに示している者を参照すること。“IdP“へのアカウントの新規作成を行う際にまず、“IdP“に管理者ユーザーで認証を行いアカウント作成の権限が付与されているトークンを取得する必要がある。次に取得したトークンを利用して“IdP“のWebAPIを活用して登録するアカウント情報をPOSTする。この際に必須となるBodyの情報は、“firstName“、“lastName“、“enabled“、“username“であり、オプションとして“email“を送信する。実行後、実行結果として“IdP“からHTTP Status 201が返却されると管理運用統合プレーンに対してSSH上で実行完了を返却することで実行した一連の流れを修了とする。ここで201以外のレスポンスが返却された場合は、管理運用統合プレーンに対してエラーが返却され

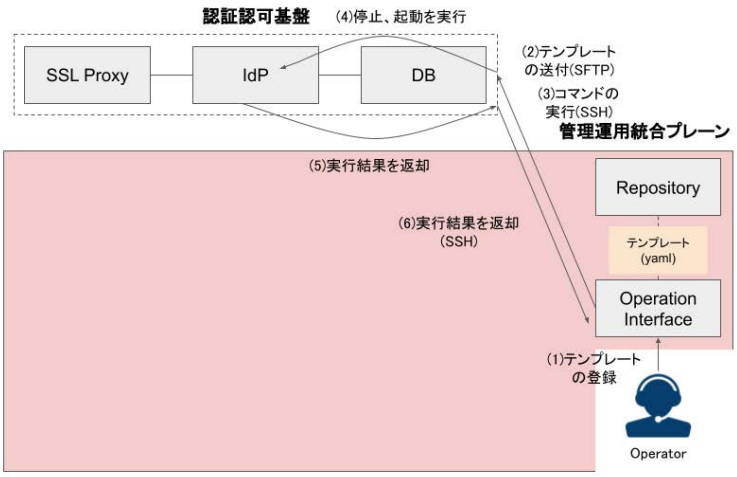

図 4.11　各モジュールへのシステム操作の一連の流れ

一連の流れが終了する。

このように認証認可基盤に対する管理運用統合プレーンの役割としては、実行する作業のコードの送信、各モジュールにおけるコードの実行、各モジュールから管理運用統合プレーンに対する実行結果の返却という一連の操作の実行となっている。

4.9.2　ログ基盤に対する管理運用

ログ基盤に対する管理運用統合プレーンの役割としては、認証認可基盤から認証認可の稼働状況のデータ、アカウントに関するアクセスログの収集、蓄積、可視化である。この役割に関する各モジュールはインターフェースとして、システムの稼働状況、システムログを JSON 形式で出力する機能を有している。またこのほかに “IdP“にはログインを行うユーザーのアクセスログを JSON で出力するインターフェースを有している。ログ基盤には、認証認可基盤から出力される全 JSON 形式データを受信する “Logging“インターフェースが用意されている。ログ基盤内では取得したデータを DB に蓄積、蓄積されているデータを “Search engine“から選択および検索し出力するインターフェースが用意されている。最終

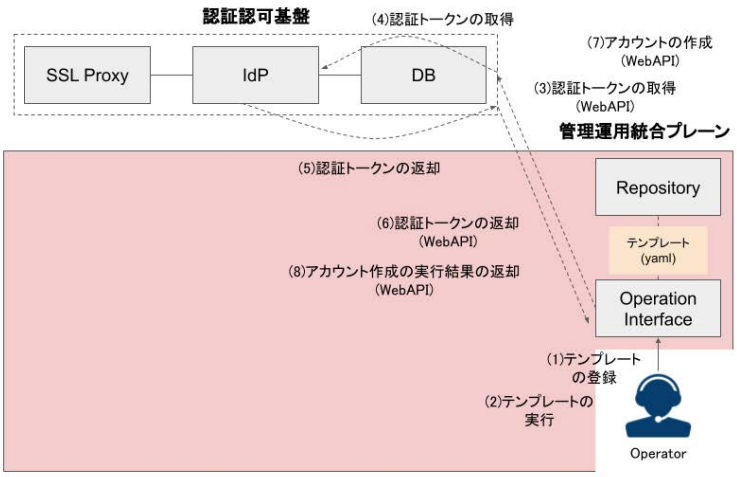

図 4.12　アカウントの新規作成の一連の流れ

的に “Dashboard“として表示される。“IdP“の稼働状況データが Dashboard に出力されるまでと “IdP“からアクセスログが生成され “Dashboard“に出力されるまでの流れを示す。

“IdP“の稼働状況データが “Dashboard“に出力されるまでについて図 4.13 に示す。まず “IdP“から出力可能な全データの項目は、付録 B に示しておりこのデータが key-value として JSON 形式で “Logging“に対して MessagePack と TCP を用いて出力される。“Logging“がデータを受信すると受信するすべてのデータを DB に蓄積する。ここまでが稼働状況データの蓄積の一連の流れとなっている。運用者が “Dashboard“から稼働状況を確認するまでを示す。運用者は Web ブラウザから “Dashboard“にアクセスしデータの検索を行う。この時の検索は “Dashboard“にアクセスした際の最初のデータの読み込みも含んでいる。検索が実行されると “Search engine“に対して WebAPI で GET を行い “Logging“を通して “DB“にクエリを送ることで、指定した情報を逆順で “DashBoard“まで返却する。これにより稼働状況データを “Dashboard“に送信している。また運用者の操作をトリガーとしてデータを呼び出す以外に、“Dashboard“から特定のデータに対して Trap を行うことも可能である。トラップの手法としては先ほどの運用者が行った検索の作業を Trap で定義したデータに対して単位時間で継続的に実行させることによって

実現している。現段階では実装していないがこのトラップをメールや Webhook を活用して特定のコミュニケーションツールに出力することで異常検知時の通知も可能である。

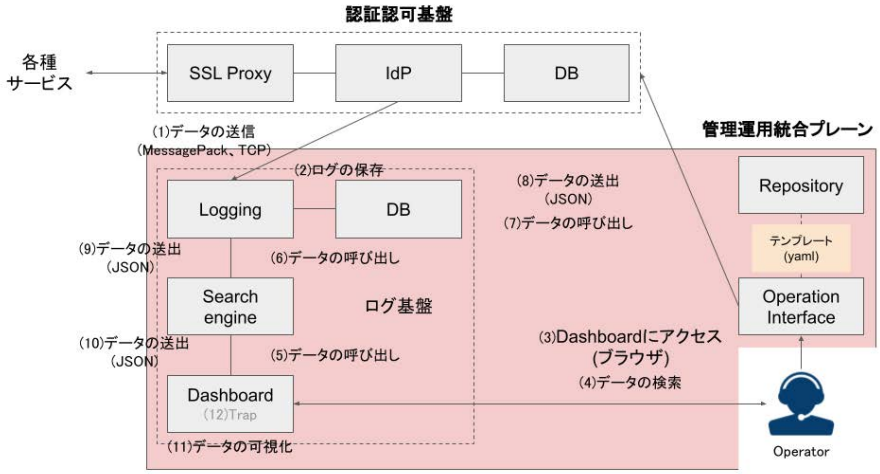

図 4.13　稼働状況データの Dashboard 出力の一連の流れ

次に “IdP“アクセスログが生成され “Dashboard“に表示されるまでについて図 4.14 に示す。まずユーザーがログイン処理を行うと “IdP“で付録 D で示している JSON 形式でアクセスログが生成される。アクセスログが生成されると同時に “Logging“に対して MessagePack と TCP を用いて出力される。“Logging“がデータを受信すると受信するすべてのデータを DB に蓄積する。ここまでがアクセスログの蓄積の一連の流れとなっている。運用者が “Dashboard“から可視化されたアクセスログを確認するまでを示す。運用者は Web ブラウザから “Dashboard“にアクセスしデータの検索を行う。この時の検索は “Dashboard“にアクセスした際の最初のデータの読み込みも含んでいる。検索が実行されると “Search engine“に対して WebAPI で GET を行い “Logging“を通して “DB“にクエリを送ることで、指定した情報を逆順で “DashBoard“まで返却する。次に返却された JSON 形式のデータを可視化されている送信元 IP、GeoIP、アクセス総数、info 等に取り出し、グラフや地図上にプロットを行っている。これによりアクセスログを “Dashboard“に送信している。また稼働状況データと同様に、“Dashboard“から特定のデータ

に対して Trap を行い、Trap で定義したデータに対して単位時間で継続的に実行させることで異常検知時の通知も可能である。

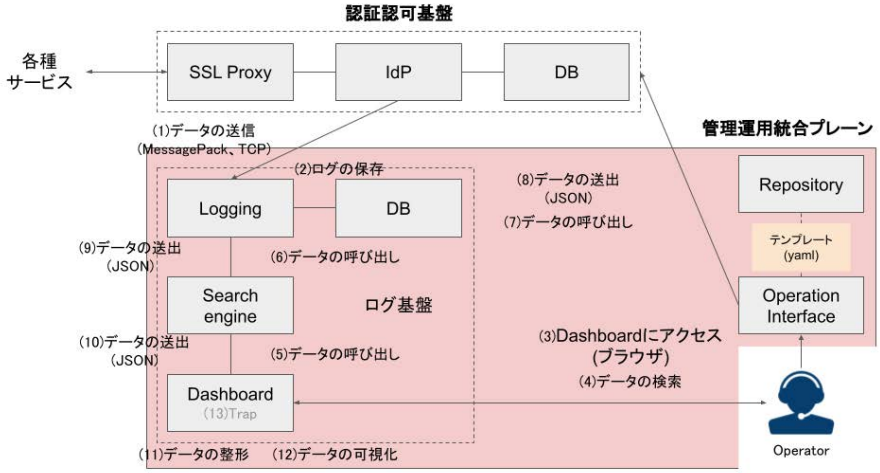

図 4.14　アクセスログの Dashboard 出力の一連の流れ

第 5 章
従来のアーキテクチャでの本提案の実現の可否

5.1. 管理運用統合プレーンアーキテクチャ実装の前提

実践的な認証認可基盤の実装にあたり、認証認可基盤の構成要素とソフトウェアの選定について述べる。

5.1.1 活用するソフトウェアの選定方法

本研究の目的は実践的に認証認可基盤を継続的に管理運用することが可能となることである。そのために利用可能なソフトウェアについては積極的に活用し、足りない部分については実装し公開する方針を取る。この手法は、CNCF [94] を代表とするオープンソースソフトウェアの共同開発に見られるように世の中の潮流となっており、日本国内においても、デジタル庁情報システム調達改革検討会 [95] や経済産業省 [96] においてもオープンソースソフトウェアの利活用について肯定的な方針となっている。この方針により安定的にソフトウェアの開発、提供、活用を実現できる。

次にソフトウェアの選定基準を示す。本研究の成果として最終的に様々な組織において認証認可基盤を管理運用することを想定しているため、ソフトウェアライセンスとソフトウェアの開発活発度を軸に利用するソフトウェアを選択している。そのため原則、一定の条件を満たすことで商用、改修、二次配布を行うことが可能なライセンスを用いることとしている。ソフトウェアライセンスの詳細については後述するが、これらを満たすライセンスとして本研究では、Apache 2.0

Licence、MIT License で作成されているソフトウェアを採用している。次にソフトウェアの開発活発度について示す。当然ではあるがソフトウェア開発が活発であり、尚且つ様々な議論が交わされ意見がコードに反映されるソフトウェアは高い安定性と品質を担保されていることが高い。代表的な例としては、Linux kernel [97] が挙げられ、Ubuntu [98]、Red Hat Enterprise Linux [99] を代表とする俗に言う Linux OS によって様々なインターネットを活用する製品に商標、無償問わず活用されている。このソフトウェアの開発活発度を測るための基準としては、社会的な注目度、開発頻度、コミュニケーションの活発度を軸として判断する。本研究においては、Black Duck Open Hub [100] を利用することで図 5.1 で示すような情報をもとに判断した。代表的な評価項目としては、

- Security Confidence Index
- Vulnerability Exposure Index
- Lines of Code
- Commits per Month
- Contributors per Month
- Lines of Code

である。ただし今回の評価の範囲外となる項目としては、意図的にコミュニティ内で脆弱性を生み出す行為 [101] [102] にあげられるような故意に発生させるインシデントについては対応できないものとする。

5.1.2 ソフトウェアライセンスと成果物の取り扱い

利用するソフトウェアライセンスについて述べる。本研究において利用するライセンスは、一定の条件を満たすことで商用、改修、二次配布を行うことが可能な Apache 2.0 Licence、MIT License である。これは本研究によって生まれた認証認可基盤および管理運用統合プレーンを一般利用をしやすい形式で公開すること

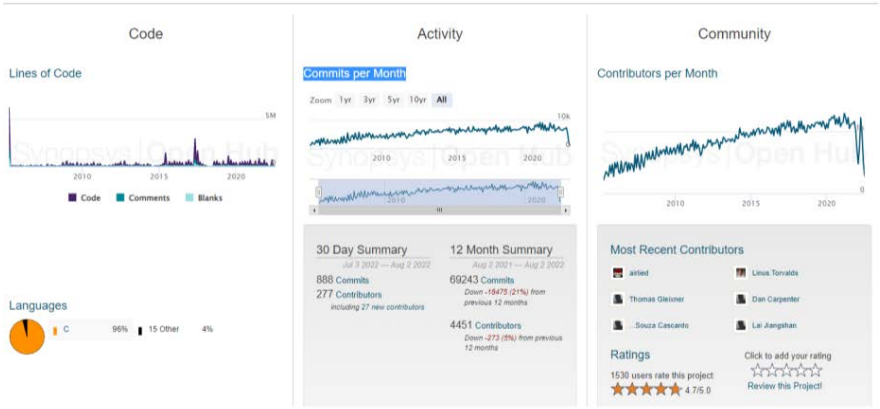

図 5.1 OpenHub の活用例:Linux kernel を例に

を目的としているためである。またこれに伴って本研究で作成したプログラムおよび IaC による各種コード郡についても、Apache 2.0 Licence および MIT License を適用して Github にて一般に公開している[1]。本成果のメンテナンスについては Issue 管理によって対応する。また利用している各種ソフトウェアがアーカイブおよびメンテナンス外となる、またライセンスが変更され本研究における前提から逸脱する場合についてはソフトウェアの再選定、認証認可基盤および管理運用統合プレーンへの影響調査を行い適切に更新していくものとする。

5.2. 認証認可基盤の構成要素

管理運用統合プレーンを実現するにあたり本研究で必要となる認証認可基盤の構成要素について、認証認可基盤の構築、管理、運用の要素に分けて示す。

1 Github - daiyak : `https://github.com/daiyak`

5.2.1 構築についての要件と実装

認証認可基盤の構成と動作環境

本研究で構築した認証認可基盤の構成について図 5.2 および表 5.1 に示す。アーキテクチャの詳細およびバージョンについては付録 A に記載する。

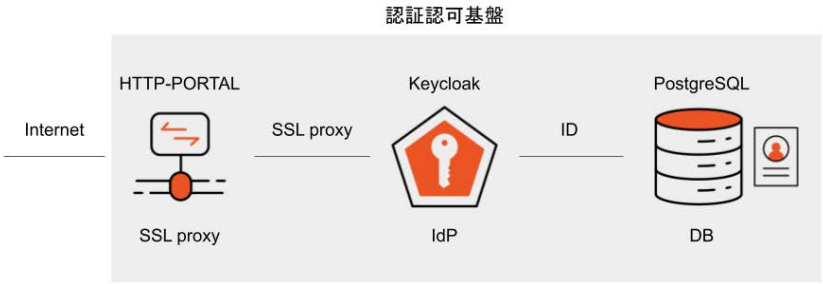

図 5.2 認証認可基盤の構成

表 5.1 認証認可基盤構成要素一覧

ソフトウェア名	機能	ライセンス
Keycloak [103]	IdP	Apache-2.0 license
PostgreSQL [104]	DB	PostgreSQL License (similar to the MIT licenses)
HTTPS-PORTAL [105]	SSL proxy	MIT license

次に動作環境について示す。本認証認可基盤の推奨動作環境については、Ubuntu、RHEL、Debian であり安定起動することを確認している。ただし検証を行ってはいないが構成上では、Linux distribution 下であれば動作すると想定している。またハードウェア要件については、導入する組織の利用者数やアクセス数によって大きくことなるため言及はしない。ただし、ハードウェア要件が足りていない状態については本研究における管理運用統合プレーンで確認することが可能となっている。

各システムのモジュール化

認証認可基盤を構成している各種システムのモジュール化を行う。ここでいうモジュール化とは、2.3 節で問題点として挙げた、ソフトウェア同士の依存関係を軽減するために各ソフトウェア毎に分離することを表している。実践的な管理運用を実現するためにはシステムの状態を把握する必要がある。そのためソフトウェア毎での正確な異常検知が必要となる。2.3 節で示した通り、認証認可機能は全体が単一であり様々なソフトウェアが複雑な依存関係を持つように構築が行われてる場合、システムの異常が発生した場合に原因箇所を特定することが困難である。これはマシン上にどのソフトウェアが動作しているのか、マシン上から大量のログやプロセスを検索し調査するためには管理運用の技能やノウハウが必要となることに起因しているためである。そこで本研究では図 5.3 に示すようなソフトウェア毎にシステムとして分離したアーキテクチャをモジュール化を用いて設計しデータプレーンおよび制御プレーンで関連付け、これらを最終的に管理運用統合プレーンで管理、制御することを可能とする。

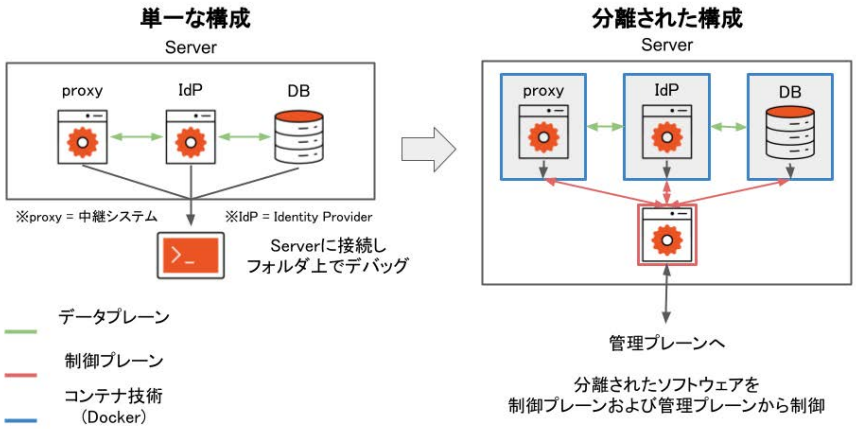

図 5.3 認証認可システムのモジュール化のイメージ

実際に構築した認証認可基盤のモジュール化とアーキテクチャについて示す。本研究では Docker [70] および docker-compose を利用しソフトウェア毎にコンテ

ナ化することで構築している。図 5.4 にコンテナ化した認証認可基盤のイメージを示す。

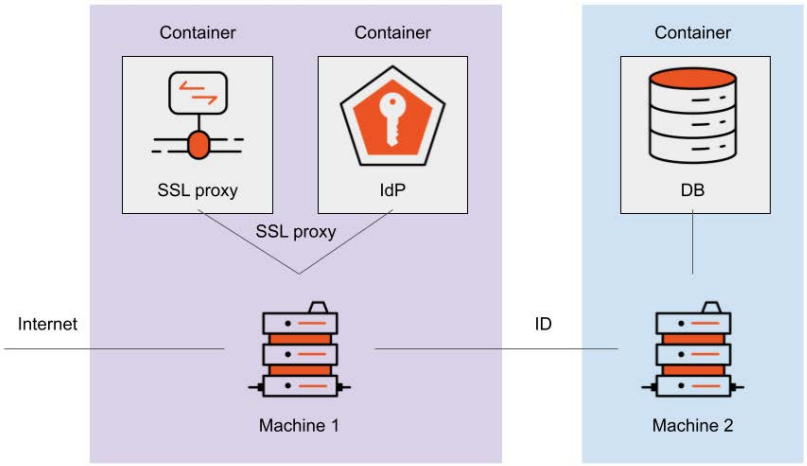

図 5.4 コンテナ化のイメージ

これにより、ソフトウェアレベルでの障害検知および障害発生時のソフトウェアの再構築などの管理運用を容易に個別に行うことが可能となり、ソフトウェア毎の状態、例えば、どのようなバージョンのどのようなソフトウェアが利用されているのか、利用しているポート情報、利用しているストレージ等を参照することが可能となる。詳細について次に述べる。

構築するための設計図と IaC

Docker を利用する理由として、ソフトウェアの分離以外にも構築するための設計図を管理フレームで管理し参照および再利用するためである。そのために認証認可基盤全体の設計図およびソフトウェア構成を手順書ではなく IaC の考え方をもとに code で表現している。例を付録 A に示す。これによりシステム構築を設計図の code を実行するだけで行うことでき、元来の手順書を読みすべてのコマンドを手入力して随時実行していくことによる更新作業時間の短縮、打ち間違いや実行されたことを確認してからコマンドを入力するなどの手順書からは読み取りに

くい文脈によるヒューマンエラーの減少、冪等性のある認証認可基盤の構築からこれまで構築にかかっていた負担を減らすことを可能にしている。

5.2.2 管理についての要件と実装

認証認可基盤を管理するためのシステムの状態および各種ログ取得、ログ基盤についての全体像を図 5.5 に構成要素を表 5.2 に示す。また管理のために必要となる要素について次に述べる。またそれぞれの構成要素の詳細については次に示す。

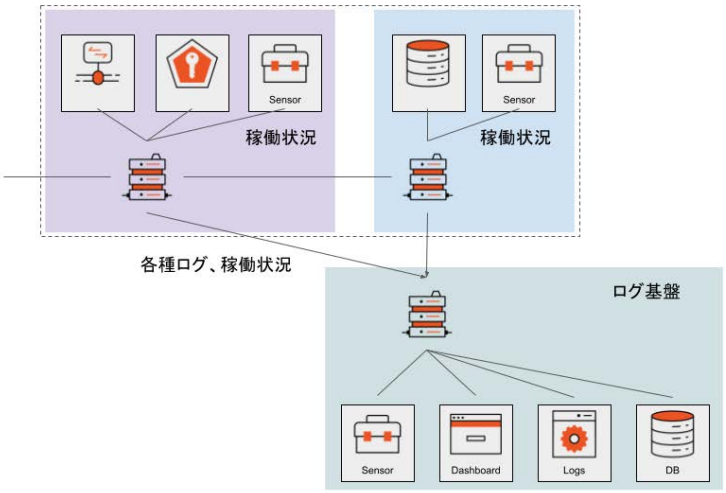

図 5.5 認証認可基盤の管理の全体像

表 5.2 ログ取得およびログ基盤要素一覧

ソフトウェア名	機能	ライセンス
Elasticsearch [74]	Search engine	Elastic License 2.0 (similar to the Apache-2.0 licenses, Commercial prohibition)
Kibana [106]	Dashboard	Elastic License 2.0 (similar to the Apache-2.0 licenses, Commercial prohibition)
Metricbeat [107]	Sensor	Apache-2.0 license
Fluentd [108]	Data collector	Apache-2.0 license

システム状態の把握

認証認可基盤の状態を把握するために、各マシンの状態を把握するのではなく動作しているソフトウェアのコンテナ単位で情報を収集している。

まずコンテナの稼働状況の取得について述べる。コンテナの稼働状況は各コンテナ内にセンサーを設置することで付録 B に示している各種情報を Metricbeat [107] を用いることで取得している。ここで取得した情報はリアルタイムで後述するログ基盤に json 形式 [109] で転送され、管理運用統合プレーンで取り扱うためにデータの整形および保存されるとともに、ダッシュボード上で認証認可基盤の状態管理を行うために活用される。本研究で構築する認証認可基盤においては取得するデータ量は圧縮しない状態で 1 日あたり 450MByte 程度となっている。

次にシステムログの取得について述べる。ここでいうシステムログはあくまでもソフトウェア単位のログであり、デプロイしているマシン自体のシステムログではないことに注意が必要である。システムログの取得は Docker [70] の logging driver 機能 [110] を用いて後述するログ基盤に json 形式で転送され、管理運用統合プレーンで取り扱うためにデータの整形および保存され、ダッシュボード上で認証認可基盤の状態管理、セキュリティチェックを行うために活用される。本研究で構築する認証認可基盤においては取得するデータ量は圧縮しない状態で 1 日あたり数十 MByte 程度となると想定している。これは認証認可基盤を利用する人数やサービス数によって大きく異なるためである。

アクセスログの収集

認証認可基盤を利用するユーザーのアクセスログの取得について述べる。このデータにはユーザーが正しくログインできた場合、ログインを失敗した場合、アプリケーションへのセッションのログを収集しており、このデータを活用することで不正なアクセスやブルートフォース攻撃の検出を行う。このデータは json 形式でログ基盤に転送されダッシュボード上で認証認可基盤のセキュリティチェックを行うために活用される。

認証認可基盤で作成するアクセスログについての詳細を述べる。認証認可基盤におけるアクセスログは Keycloak にアクセスする際に付録 D に記載されている項

目を記録することで作成している。本研究で用いている Keycloak には標準でアクセスログを出力する機能を備えていない。これは設計上アクセスログを Keycloak 内独自に取り扱うことでセッションを管理する機能を有しているためである。しかしこれでは運用者が管理運用のために見なければならない項目が増えるため本研究の目的にはそぐわないため、本研究で活用可能となるログの作成と整形を行った。この際に作成した設定ファイルを付録 D に示す。具体的には Keycloak で用いている Jboss [78] の監査ログとイベントログ機能を活用し、データを json に整形したのちに Docker の標準出力にログを出力することで最終的にログ基盤に転送を行うプログラムとなっている。ただし本来であれば D に示している通りの key-value で json を作成すること望ましい、しかし Jboss-Docker 間においてこの整形を行うことができないため、message 内にすべてのログを挿入することでログ基盤への転送を実現していることに注意が必要である。

ログ基盤の構築

ここまでで述べた各種データを実際の管理運用に活用するためのログ基盤について述べる。ログ基盤の役割としては全ソフトウェアおよびコンテナの稼働状況、各種ログを一括して収集し可視化、さらに管理運用を助けるための情報を追加することである。

ログ基盤は Fluentd [108]、Elasticsearch [74]、Kibana [106] で構成されており、Fluentd でデータを収集、Elasticsearch でデータの加工及び検索、Kibana で管理運用を手助ける形式に可視化を担っている。これらは OSS の汎用ログ基盤として Analytics on AWS [111] を代表とするように様々な環境に合わせて世界中で利活用されているが、商用利用を行う際は使用料を支払わなければならないライセンスとなっている。そのため本研究でのログ基盤は商用利用することはできない。将来的には商用利用可能な OSS を利用可能なソフトウェアを活用したログ基盤を構築するオプションを追加予定である。

本研究では認証認可基盤を実践導入し管理運用をするため専用ログ基盤として調整し再構築している。その中でも認証認可基盤のアクセスログの取り扱いについて述べる。まずアクセスログの整形について述べる。実際に作成したデータテ

ンプレートは付録 D に示す。認証認可基盤のアクセスログは接続元 IP、利用した認証プロトコル、利用しているサービス、UserName、code など通常のログよりも項目が多く string で 300 文字を超える比較的大きなログとなっている。今回用いているログ基盤ではこのような大きさのログには対応していないため入力するデータテンプレートを定義する必要がある。また本研究ではアクセスログの収集で述べた通り、本来取り扱うことを想定していない形式のログが認証認可基盤から送信されてくるためデータを可視化しやすい形式に整形する必要がある。また同時に管理運用で実際に利用しやすいように数値に意味を持たせる処理を行う。本研究では message 内のデータを付録 D にあるようにデータをパースし、さらに IP アドレスを GeoIP [112] に紐づけることで接続元の地域を特定するデータへと変換を行っている。最終的に取り扱うアクセスログを図 5.6 に示す。実際に整形されたアクセスログには、数値としてではなく実際に管理運用統合プレーンで利用するための意味の持つデータ (geoip.city_name Minato-ku, geoip.city_name Minato-ku) が出力されていることがわかる。

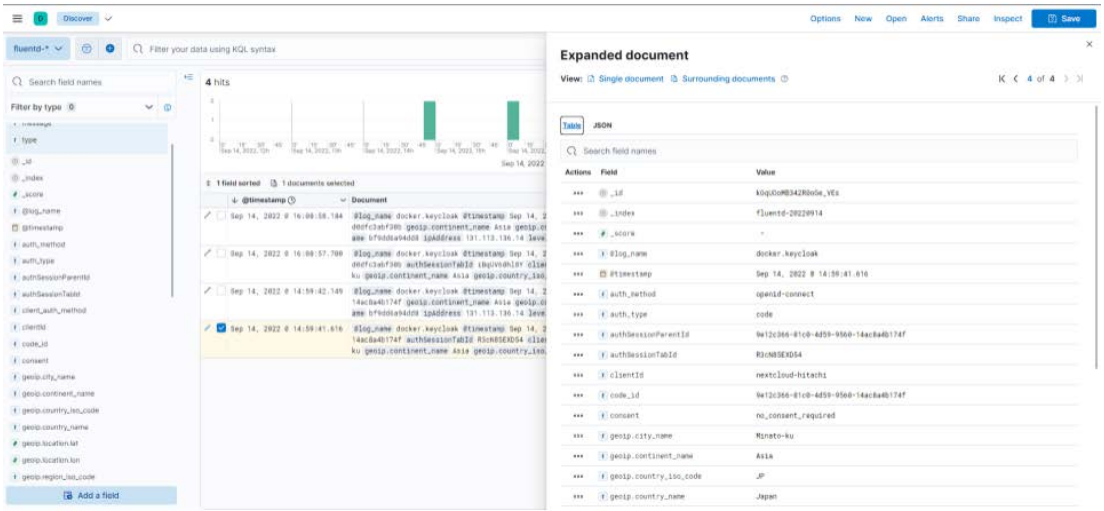

図 5.6 本研究で整形した実際のアクセスログ

5.2.3 運用についての要件と実装

認証認可基盤の運用について述べる。認証認可基盤の運用は後述する管理運用統合プレーンから行うことを想定している。具体的な運用の内容としては、認証認可基盤に利用されるソフトウェアの運用、Digital Identity の運用に分類して行う。まず認証認可基盤に利用されるソフトウェアの管理運用について述べる。この運用で行う行動としては利用しているソフトウェアに何かしたらのエラーが発生した場合の対処である。発生しうる対応例をいくつか挙げる。

運用例 1： ソフトウェアのクラッシュによる IdP の停止

ソフトウェアのクラッシュが発生した場合、運用者はログから原因を調査したのちにシステムを再起動する必要がある。その時の一連の流れは、

- IdP の停止を検知
- システムの稼働状況から IdP が停止していることを確認
- システムログからアクセス数の一時的な増加や例外処理によるクラッシュだと判断
- 対象マシンにログインし、IdP が停止していることを確認
- 停止しているシステムを Dockerfile から再起動
- IdP が正常作動していることを確認するテストを実施、復旧

となる。

運用例 2： 証明書の期限切れによる IdP の機能停止

証明書の期限切れによる IdP の機能が停止した場合、運用者はログからなぜ機能が停止しているのかの原因を調査したのちに、証明書の期限切れであることを導き出し、証明書を更新しシステムを再起動する必要がある。その時の一連の流れは、

- IdP の機能停止が利用者から報告
- システムの稼働状況から IdP が正常に動作していることを確認
- システムログから IdP で証明書が検証できていないことを確認
- 対象マシンにログインし、SSL proxy を停止
- 証明書を更新
- 停止しているシステムを Dockerfile から再起動
- IdP が正常作動していることを確認するテストを実施、復旧

となる。

基本的な一連の流れとしては状況把握、システムの再構築、動作確認となっており、確認するべき項目や実行しなればならないコマンド等は違うものの類似した行動を取る必要がある。また例に挙げた認諸認可を行うことができないという同じ事象にもかかわらず原因や行うべき運用が異なっている。

次に、Digital Identity の運用について述べる。この運用で行う行動としてはアカウントやそれに伴う Digital Identity の操作、不正ログインなどのセキュリティ問題によるアカウントの停止等のアカウントマネジメントである。発生しうる対応例をいくつか挙げる。

運用例 3 : アカウントの新規作成

アカウントを IdP に新規作成を行う場合、IdP にアカウントを作成し、一時的なパスワードを付与しユーザーがログインした際にパスワードを変更させるという運用になる。その一連の流れは、

- IdP に管理者としてアクセス
- GUI を操作してアカウントを作成、パスワードの付与
- GUI を操作してアカウントポリシーを変更し、はじめてのログインの際にパスワード変更の処理を追加

- 人数分上記操作を繰り返す
- アカウントリストと照らし合わせて、アカウントが正しく作成されたことを確認

となる。

運用例 4：不正ログインによる特定アカウントの停止

アカウントを IdP に新規作成を行う場合、IdP にアカウントを作成し、一時的なパスワードを付与しユーザーがログインした際にパスワードを変更させるという運用になる。その一連の流れは、

- 不正ログインを検知
- アクセスログから不審なアクセス元から特定のアカウントでログインを複数回繰り返していることを検知
- アクセスログからログインが成功していることを確認
- IdP に管理者としてアクセス
- GUI を操作してアカウントを停止
- アカウントが停止していることを確認

となる。

Digital Identity の操作については、IdP を操作し実行した操作が正常に行われたかを確認するという流れになっており、アカウントマネジメントについて、状況把握、アカウントの操作、アカウントの状況確認となっている。

このように認証認可基盤の運用については、先に述べたログ基盤のログ一覧からの状況把握、コマンド操作による特定のソフトウェアに対しての操作および環境の再構築、GUI の操作によるアカウントマネジメントを行うことが可能となっている。

5.3. 管理運用統合プレーンの全体構成

認証認可技術の課題を整理し、これを実現するための認証認可基盤のアーキテクチャの設計及び構築を行った。本章ではこの認証認可基盤を実践的に利用していくための管理運用統合プレーンの実装および構成について述べる。

まず管理運用統合プレーンは本研究の目的である認証認可基盤を実践的に活用するために必要となる管理運用を補助するための役割を担うシステムである。管理運用統合プレーン全体の構成について図 5.7 に示す。

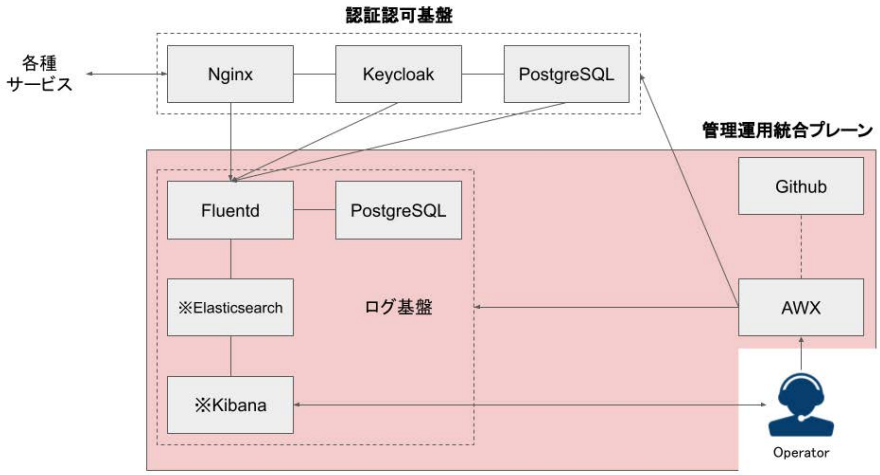

図 5.7 管理運用統合プレーン全体の構成

管理運用統合プレーンは役割毎に 3 種類の機能によって構成されており、本研究ではそれぞれを Log Dashboard、Mgmt Dashboard、IaC Repository と呼ぶ。

管理運用統合プレーンを用いた管理運用の一連の流れを図 5.7 に沿って説明する。事前準備として管理運用統合プレーンで実際に利用する管理運用をコード化した IaC を IaC Repository に登録し、管理運用統合プレーンから管理対象となるマシンの IP アドレスおよび対象となるソフトウェアが動作しているポート番号、SSH に利用するアクセス情報を Mgmt Dashboard へ登録を行う。これにより管理運用統合プレーンを利用することが可能になる。まず、(1)Log Dashboard から管理運用に必要となる情報を取得する。Log Dashboard はログ基盤と紐づい

ており、システムの状態のアラートおよびアクセスログや脅威を視覚的に確認することが可能になっている。ここで得た情報をもとに実際の管理運用を (2)Mgmt Dashboard を用いて実施する。Mgmt Dashboard では管理運用に必要となる作業を (3)IaC Repository からテンプレートとして呼び出し、必要となる環境変数を指定することで (4) 対象となるホストおよびマシンに対して実行し、実行結果を Mgmt Dashboard に返すことで終了とする。

この際に (2)(3) で利用したいテンプレートおよび IaC が存在しない場合については直接ホストやソフトウェアに対してコマンド実行等の適切な作業が必要となるが、この行った作業を自ら IaC として定義することで同様の作業をテンプレート化することが可能である。この一連の流れを繰り返すことにより実行すべき管理運用が IaC として定義され続け、利用すればするほど管理運用のコストを抑えることが見込める。

次にそれぞれの機能の役割と構成について述べる。

5.3.1 Log Dashboard の役割と構成

Log Dashboard の役割と構成について述べる。Log Dashboard の役割はログ基盤に蓄積されている情報を可視化する、また各種システムの状態に対して閾値を定義することでアラートを出力することである。実際の Log Dashboard を図 5.8、図 5.9 に示す。

Log Dashboard は、Kibana [106] を活用して認証認可基盤用の監視ダッシュボートとして実装したものである。本ダッシュボードは、アクセスログに関しては、全アクセスログ総数、ログイン元 IP アドレスと関連するアクセスログの総数、アクセスログのタイプと総数、ログイン先のクライアントとアクセス回数、上位 30 個のアクセスログの詳細を指定した単位時間でソートすることが可能なテンプレートを用意している。システムの状態に関しては、現在管理対象になっているコンテナの概要、CPU 使用状況、MEM 使用状況、ネットワーク I/O 状況リアルタイムおよび指定した単位時間で検索することが可能なテンプレートを用意している。これらのダッシュボードは必要な情報があれば自由に編集可能であり、json 形式でテンプレートをインポートおよびエクスポートすることもできる。

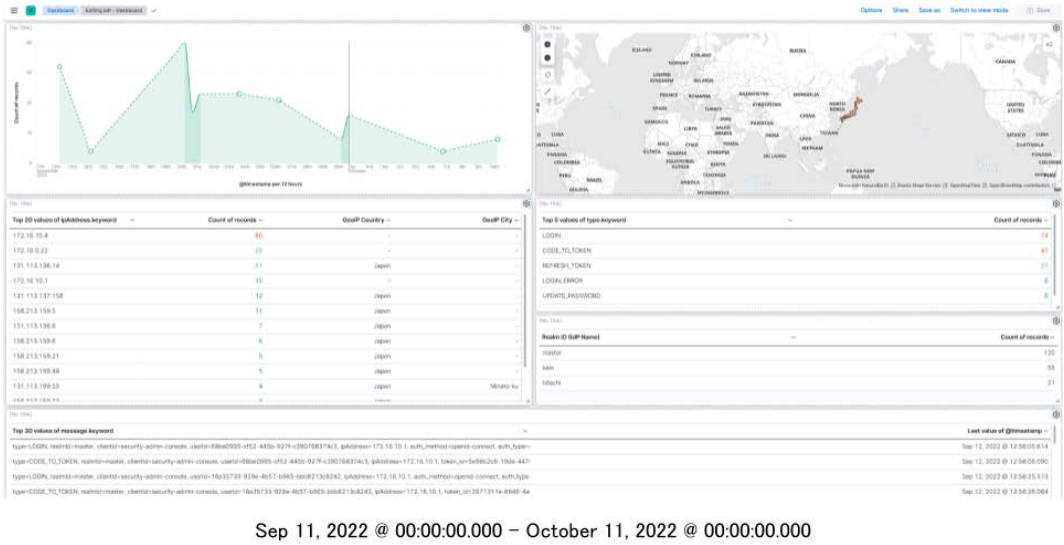

図 5.8 Log Dashboard によるアクセスログの可視化

現状の管理運用統合プレーンのアラートは 3 段階で表現しており、システムの状態に関しては数値全体の 90%で RED、80%で YELLOW、通常状態を GREEN で可視化しダッシュボード上に通知している。同様にログに関しては同じ種類のログが単位時間あたりに 80 件を超えたら RED、40 件を超えたら YELLOW、それ以外を GREEN で可視化しダッシュボード上に通知している。この通知に関してはダッシュボード上だけではなく、webhook を用いることで Slack や Discord へ、SMTP を活用できる環境があればメールとして通知することも可能である。

また利用する組織によっては、ログの保存と管理を厳格なポリシーで行う必要がある場合がある。本管理運用統合プレーンではログの保存期間を json 形式で設定することが可能であり、デフォルトとして刑事訴訟法 第百九十七条[2]およびサイバー犯罪に関する条約 第十六条[3]に対応して、すぐに検索できる非圧縮の形式

2 刑事訴訟法 第百九十七条:https://elaws.e-gov.go.jp/document?lawid=323AC0000000131

3 サイバー犯罪に関する条約 第十六条:https://www.mofa.go.jp/mofaj/gaiko/treaty/treaty159_4.html

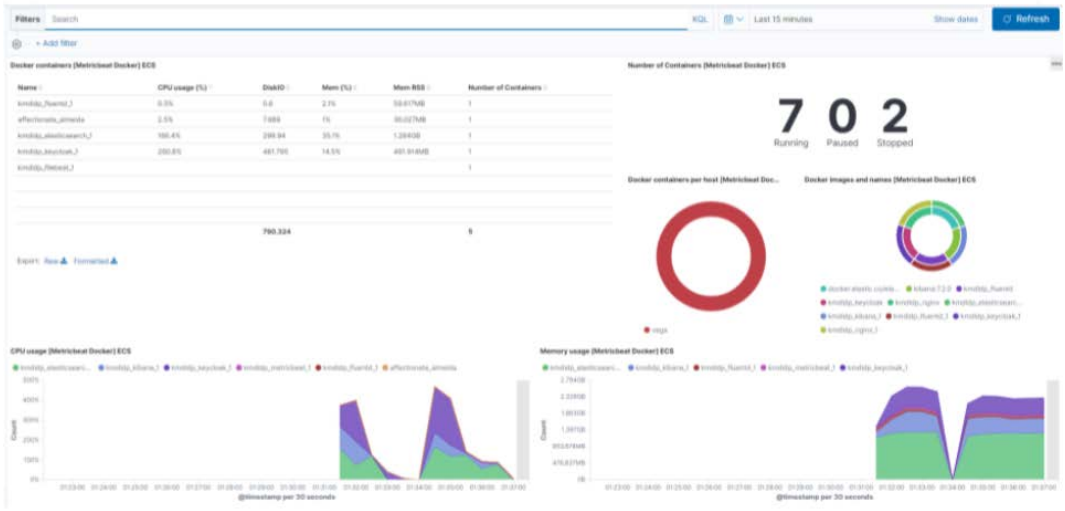

図 5.9 Log Dashboard によるシステム状態の可視化

で 90 日、データ量を抑えて圧縮した形式で 365 日保存を行うライフサイクルポリシーを作成し適用している。

5.3.2 Mgmt Dashboard の役割と構成

Mgmt Dashboard の役割と構成について述べる。Mgmt Dashboard の役割は管理運用対象となるマシンおよびソフトウェアに対して IaC Repository に保存されているコードから運用者が実際に行う作業をテンプレート化することでボタン一つで任意の作業を実行することである。実際の Mgmt Dashboard を図 5.10 に示す。

Mgmt Dashboard は、AWX [113] を活用して認証認可基盤用の管理運用ダッシュボートとして利用しているものである。基本的な仕組みとしては、Ansible [82] と呼ばれる「プロビジョニング、構成管理、アプリケーションのデプロイメント、オーケストレーション、その他多くの IT プロセスを自動化する、オープンソースの

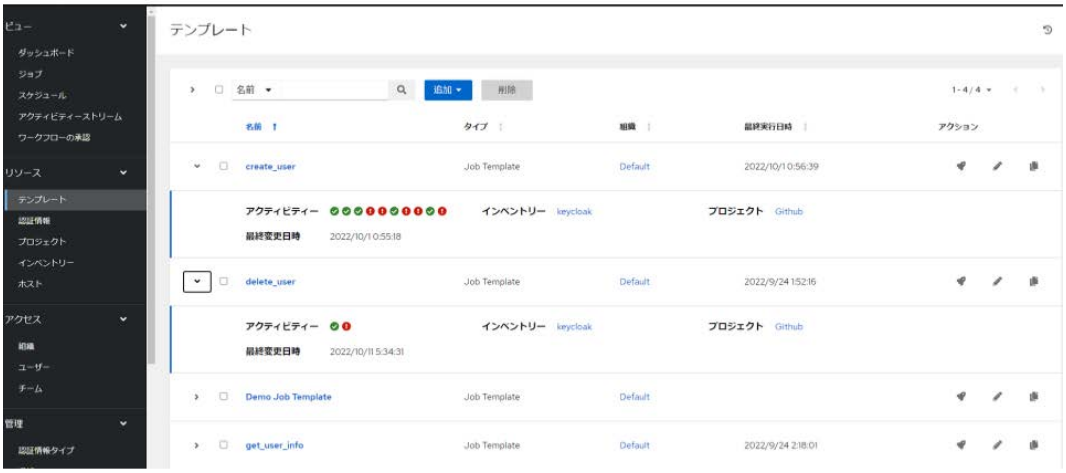

図 5.10 Mgmt Dashboard による作業のテンプレート化

IT 自動化エンジン」[4]を用いて作業をテンプレートとして定義している。

本研究で取り扱うテンプレートの要件について示す。本管理運用統合プレーンで取り扱うテンプレートは、

- 認証認可基盤全体のシステムの管理運用を行う認証認可基盤運用テンプレート
- アカウント操作を行う Digital Identity 運用テンプレート

を現状採用した。認証認可基盤運用テンプレートについては、認証認可基盤を構築しているシステムを継続して管理運用していくこと目的とし運用者が直接操作する作業をサポートを行うことを目的としている。Digital Identity 運用テンプレートについては、認証認可基盤で活用および管理するアカウントの作成や削除、個々のアカウントの利用可能状態の管理の管理を行うことを目的としている。その他

4 Ansible とは:https://www.redhat.com/ja/topics/automation/learning-ansible-tutorial

にも考えられる操作としては、アカウントに付与されている認可ポリシーの操作、IdP を利用する Web サービスの管理、Digital Identity に含まれている個人情報の編集等があるが、実際に認証認可基盤を運用しフィードバックを受け、必要なものを随時追加すること、またこれらの運用テンプレートの作成方法のドキュメントを行い運用者が必要なテンプレートを作成可能とすることを今後の課題としている。

本管理運用統合プレーンで定義している作業の一例としてを付録 F をもとに説明する。このコードは、IdP に新規アカウントを作成する作業を定義しており、具体的な動作としては、Keycloak に管理者権限でログインし、REST API を活用して作成したいアカウントの情報を json 形式で PUSH することによってアカウント作成を行うという作業を示している。まずディレクトリ構造と記法については、Ansible の作法に準拠して作成しているためベストプラクティス[5]を確認すること。作業に関連するコードは、tasks と vars によって定義されており、tasks には実際に実行する作業を、vars には作業を実行する際に必要となる環境変数を定義している。環境変数を定義しておくことで、テンプレートを実行する際にダッシュボード上から直接変数を柔軟に指定することが可能となり管理運用のコストを減少させることができる。次にテンプレートを実際に実行する場所を inventory で定義する。今回の場合は Keycloak の API をダッシュボード上から利用するため localhost を指定している。最後にダッシュボード上で利用可能にするためのテンプレートファイル (ここでは、keycloak_create_user.yml) を作成する。実際に Mgmt Dashboard 上で読み込んだ本テンプレートを図 5.11 に示す。

テンプレートを実行するには、環境変数を任意のものに修正したのちに図 5.11 の下部にある起動ボタンを押すのみとなっている。実際に実行された結果を図 5.12 に示す。実行時間については作業内容によって異なるが今回の場合は 7 秒であった。

現在 Mgmt Dashboard に実装されているテンプレートの一覧とその機能について示す。これらは全て MIT license で公開しており、テンプレートの詳細な利用方法については付録に記載されている Github リポジトリを参照すること。これらのテンプレートは現在行っている管理運用統合プレーンの運用を通して今後も追加

5 Ansible ベストプラクティス:https://docs.ansible.com/ansible/2.9_ja/user_guide/playbooks_best_practices.html#directory-layout

図 5.11 作成したテンプレートを活用する例

予定あり都度公開予定である。

認証認可基盤運用テンプレート

- machine-alive
  - 指定したマシンの死活管理を ping を用いて行う
- container-start
  - 指定したコンテナを起動
- container-stop
  - 指定したコンテナを停止
- container-restart
  - 指定したコンテナを再起動
- container-rebuild
  - 指定したコンテナをリビルドして起動

認証認可ポリシー運用テンプレート

- create_user
  - 新規アカウントの作成
- delete_user
  - 指定したアカウントの削除

図 5.12 テンプレートの実行と作業結果

- `get_user_info`
  - 指定したアカウントのユーザー情報を取得
- `update_user_info`
  - 指定したアカウントの指定したユーザー情報を更新
- `enable_user`
  - 指定したアカウントをログイン可能にする
- `disable_user`
  - 指定したアカウントをログイン不可能にする
- `get_user_roles`
  - 指定したアカウントのロール情報を取得
- `update_user_roles`
  - 指定したアカウントの指定したロールに変更する
- `update_user_password`
  - 指定したアカウントの指定したパスワードに変更する

5.3.3 IaC Repository の役割と構成

IaC Repository の役割と構成について述べる。IaC Repository の役割は Mgmt Dashboard で利用するテンプレートを含む全コードを保存し管理することである。本 IaC Repository では図 5.7 に示す (3) 管理運用統合プレーン上に Gitlab を構築して管理する方法と (3.1) 外部サービスとして Github を活用する方法を提供している。ここでは Github 上で管理している例を図 5.13 に示す。

基本的な利用方法としては、まず公開しているテンプレートをダウンロードし git コマンドを活用して (3)or(3.1) に push することでテンプレートを保存する。次に Mgmt Dashboard にテンプレートを保存している Repository の登録を行う。利用する Repository 上に Mgmt Dashboard 用のアカウントを作成し、この際に発行される Personal access tokens を活用することで Mgmt Dashboard と IaC Repository の連携を行う[6]。

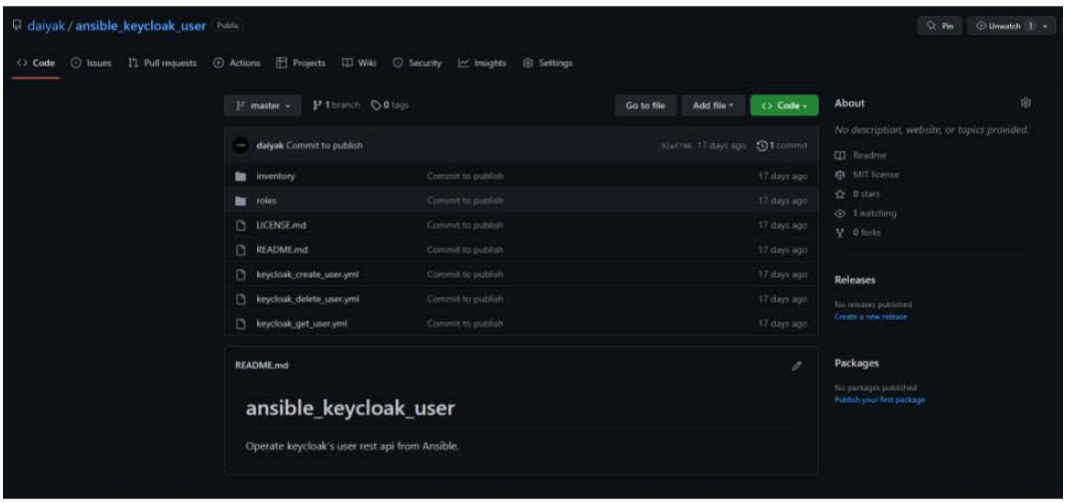

図 5.13 Github 上でのテンプレートの管理

6 Creating a personal access token:https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token

5.4. 本設計、実装と課題への適用

本章では 3.2 節で示した課題をもとに 4.7 節に設計と実装の方針を示し認証認可基盤と管理運用統合プレーンの設計および実装を行った。本節ではこれらの設計、実装と課題への適用についてまとめ、実際に課題を満たしているのかを判断するための評価方法について述べる。

まず、認証認可基盤全体のフレームワーク化についてまとめる。環境への依存が少ないシステム構成については、4.7 節で示したアーキテクチャおよび 5.2 節のモジュール化の通り、それぞれの構成要素ごとに Docker によりコンテナ化することによって実現している。また同様に再現性の高い認証認可基盤のワンストップな構築についても、構築手順をコードで表現し実行することで同様な環境を構築可能としている。これらは実際に想定される様々な環境に実際に認証認可基盤を構築することで評価を行う。継続的な管理運用が容易なシステムアーキテクチャについては、図 4.10 に示すように管理運用統合プレーンをそれぞれのシステムに対して 5.3.2 項で示した IaC を用いた管理運用テンプレートを反映可能とすることで実現している。この評価については本研究で示した認証認可基盤および管理運用統合プレーンを実際に運用し、システムの管理運用が設計通りに行われたかどうかを観察することで行う。

認証認可基盤の管理運用の簡略化についてまとめる。認証認可基盤の状況把握については、5.2.2 項に示した各構成要素からのデータの取得、5.3.1 項のログ基盤による収集と可視化を行うことで実現している。評価については、4.8 節で示した必要となるデータが目的通り可視化され要件を満たしていることを確認する。認証認可基盤の管理運用の簡略化については、5.2 節による管理しやすい認証認可基盤の構成、5.3 節による管理運用統合プレーンによる管理の簡略化と運用のテンプレート化によって実現を図る。評価については、認証認可基盤および管理運用統合プレーンをテスト環境で実際に利用してもらうことによる性能テスト、実環境における活用事例と実際に行った運用を示すことで評価とする。ユーザーアクセスとアカウントの状況把握については、4.8 節で設計したデータを 5.2.2 項の要件に合わせて認証認可基盤からアクセスログを取得し 5.3.1 項で構築するログダッシュボードを活用することによって実現する。評価については設計した要件が満たさ

れているのかを実際に検証することによって確認する。また選定した各種データの利用頻度および過不足については前述した実運用をもとに考察する。アカウントマネジメントの簡略化については、ユーザーアクセスとアカウントの状況把握を行った後に 5.3.2 項、5.3.3 項で設計した運用テンプレートを用いることで実現を図る。評価としては、5.2.3 項に挙げたこれまでの運用例と比較することにより定性的な評価を行う。

ここまでで 3.2 節で示した課題をもとに要件を満たす認証認可基盤および管理運用統合プレーンについて設計と実装を行った。また設計実装した認証認可基盤および管理運用統合プレーンのそれぞれの評価項目について述べた。次にこれらの評価結果について 6 章に示す。

第 6 章
管理運用統合プレーンアーキテクチャの実証評価

本章では本提案の認証認可基盤および管理運用統合プレーンアーキテクチャが本研究の目的を満たしているかについて、実際に認証認可基盤の実現のための管理運用統合プレーンアーキテクチャを導入し、活用、管理運用することによって実証評価を行う。

6.1. 本提案アーキテクチャの評価

本提案の認証認可基盤および管理運用統合プレーンアーキテクチャが本研究の目的を満たしているかについて、実際に認証認可基盤の実現のための管理運用統合プレーンアーキテクチャを導入し、活用、管理運用することによって実証評価を行う。

評価する項目としては、4.4 節で示した認証認可基盤および管理運用統合プレーンアーキテクチャの要件を満たし、本提案アーキテクチャの目的である組織の認証認可ポリシーとシステムポリシーを守り、判断する。そのための評価項目と評価軸について述べる。まず認証認可基盤については、

- 自由なソフトウェアの選択が可能なアークテクチャ
- 構築環境への依存が少ないアーキテクチャ
- ソフトウェアに依存せずに管理情報を取得可能なアーキテクチャ

を満たすことで組織のシステムポリシーが反映可能でかつ構築が可能な認証認可基盤であることを示す。そのために評価としては、実際に異なるユースケースと環境に対して本提案の認証認可基盤および管理運用統合プレーンアーキテクチャを導入し、サービスが利用可能かつ認証認可基盤を運用可能な状態となっているかを実証評価することで評価とする。このとき運用者からヒアリングを行い、認証認可基盤および管理運用統合プレーンアーキテクチャの改善と要件を満たしているかを判断する。

次に管理運用統合プレーンについては、

- 認証認可基盤からの管理情報によるシステムの状況把握
- 認証認可基盤からのアクセスログによるアカウントの状況把握
- 認証認可基盤に対する認証、認可、システムの運用の実施とポリシーの適用

を満たすことで認証認可ポリシーの適応と認証認可基盤の継続的な管理運用を行うことが可能なアーキテクチャとなっていることを示す。そのための評価としては、実際に認証認可基盤および管理運用統合プレーンアーキテクチャを導入した環境において認証認可基盤の運用を実施した。実施の中で得られた実証と従来の管理手法と比較してすること本提案アーキテクチャが実現され、管理運用の統合を実現できているかを検証する。この際の、

- 発生した事象と運用を伴ったケース
- 実際に行った運用
- 行った運用に対する運用者のコメント

をまとめ、4.4 節の要件を満たし、4 章で設計したアーキテクチャと比較を行うことで、管理運用統合プレーンとアーキテクチャが実現されいるかを示す。

6.2. 認証認可基盤および管理運用統合プレーンの構築について

認証認可基盤および管理運用統合プレーンの構築についての評価を行う。評価方法としては、被験者に実際に認証認可基盤および管理運用統合プレーンを用意した環境に実際に構築してもらうことで、実際に構築ができるのか、また構築時に問題は発生するのかを確認することで行った。また被験者へのインタビューを行うことで課題を洗い出すことで修正を行った。被験者については 6.2.4 項に示す。

6.2.1 既存の手法による認証認可基盤の構築

まず評価の比較対象となる既存の手法による認証認可基盤の構築について示す。ここでの構築は 5.2 節で示した本研究で利用している認証認可基盤のアーキテクチャを用いる。構築する環境は Ubuntu 22.04 Server としている。大まかな構築手順としては、

1. 構築する環境に合わせて、各種ソフトウェアの導入方法の確認および導入する環境を整備する
2. 自身の環境に合わせて適切な導入方法を選択し、SSL Proxy を構築する
3. 自身の環境に合わせて適切な導入方法を選択し、IdP を構築する
4. 自身の環境に合わせて適切な導入方法を選択し、DB を構築する
5. IdP-DB 間の接続性を確認し正常に動作するかのテストを行う
6. SSL Proxy-IdP 間の接続性を確認し正常に動作するかのテストを行う
7. IdP が正常に動作するのかのテストを行う

の大きく 7 ステップの手順となっている。次に各手順における構築時の負担となりうる事象について述べる。まず構築ドキュメントの読み間違いや解釈の勘違いによる構築時のエラーの発生である。これは構築者は人間であるためミスをして

しまうという前提とすると構築の際の冪等性が担保できていないため発生する事象である。次に 5、6、7 で挙げた動作テストにおいて正常な動作が確認できない場合である。この際に構築者は各ソフトウェアのログを読み原因を特定するという負担の大きな作業が必要になる。これは様々な原因が考えられるためログを確認しなければ問題を特定することは困難であるが、各機能同士の接続性が担保されていないため発生する事象である。

この構築者の負担と冪等性の問題が解決されているのかについて、本研究における認証認可基盤全体のフレームワークを活用して実際に同様の環境で構築を行い比較評価を行う。

6.2.2 構築のための前提と環境整備

本認証認可基盤および管理運用統合プレーン構築のための前提について述べる。本システムは、“docker“、“docker-compose“がインストールされていることが前提となっている。前提となる環境を整備することについては本研究の範囲外であり、Docker [70] のベストプレクティスを参照し自身の環境を整備するものとする。とはいえ、公開してい本認証認可基盤および管理運用統合プレーンに README として環境整備方法を記載している。また構築されるマシンのハードウェアの構成と使用するマシン台数、ネットワークアーキテクチャについても本研究では議論の対象外としている。しかしこれについても考えうるアーキテクチャにできる限り対応を行いソースコードに例となる全体の構成を記載することで対応を行っている。

本稿の評価においては本研究が活用される環境の中でもっとも一般的であると思われるアーキテクチャで行っている。

6.2.3 構築テストで用いた環境

今回構築に利用したマシン環境は以下の通りである。

- OS:Ubuntu 22.04 Server

- CPU:4Core
- Mem:8GB
- Disk:32GB

この環境を図 6.1 に示すアーキテクチャで 3 つ用意し、それぞれに認証認可基盤および管理運用統合プレーンの構築テストを行った。ネットワークについては、(1) が対外アクセスがあるグローバルなネットワーク、点線をまたぐ矢印が組織内のプライベートなネットワーク、点線内が Docker 内のローカルネットワークとなっている。

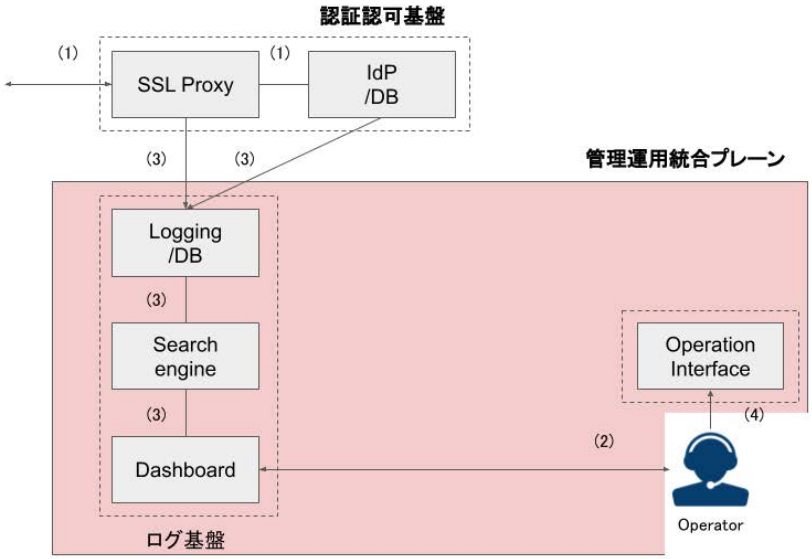

図 6.1　構築テストで用いた環境

このアーキテクチャで実際に認証認可基盤および管理運用統合プレーンが 4.6 節、4.7、5.2 節の要件に沿う形で構築を行うことが可能となるのかの構築テストを行った。

6.2.4 被験者

今回実際に認証認可基盤および管理運用統合プレーンの構築を行った被験者の情報を示す。

- バックグラウンド: 理工学系後期博士課程在籍
- 経験: システムの継続的な管理運用の経験なし、認証認可基盤の実装、構築経験なし、docker の利用経験なし
- 有している技能: ディレクトリ移動やファイル操作などの Linux の基本的なコマンド操作が可能

後述する評価以前に本認証認可基盤構築で利用しているソフトウェアの利用経験、事前知識がないため、評価前に実際に公開している本研究の認証認可基盤および管理運用統合プレーンのソースコードに含まれている README の構築手順に従って構築を行ってもらうこととした。

6.2.5 プロトタイプ 1: 最小手順での構築

プロトタイプ 1 として、3.2 節のターゲットに対して、再現性の高い認証認可基盤のワンストップな構築を実現するために構築手順を最小限にすることを目的として構築を行った。

構築方法と手順

まず認証認可基盤の構築方法と手順について述べる。なお前述した環境整備およびソースコードのダウンロード、展開については記述しない。構築時のディレクトリはダウンロードし展開したソースコードのトップディレクトリとする。手順と実際の操作については以下の通り。

1. 自身の組織に合わせて、初期ユーザー、パスワード、データベースのパスワード、証明書のドメイン名などの環境変数をソースコードから確認し修正

2. 構成手順が記載された docker-compose.yml を起動
   コマンド:

```
$ docker-compose up -d --build
```

   出力:

```
 daiya@sandboxidp-portal: /idp$ docker-compose up -d --build
WARNING: The SSL_DOMAIN variable is not set.  Defaulting to
a blank string.
WARNING: The SSL_STAGING variable is not set.  Defaulting to
a blank string.
WARNING: The DB_ADDR variable is not set.  Defaulting to
a blank string.
WARNING: The DB_DATABASE variable is not set.  Defaulting to
a blank string.
WARNING: The DB_USER variable is not set.  Defaulting to
a blank string.
WARNING: The DB_PASSWORD variable is not set.  Defaulting to
a blank string.
Building keycloak
Sending build context to Docker daemon 9.728kB
Step 1/2 :  FROM jboss/keycloak
--->011a708d97bf
Step 2/2 :  ADD themes/welcome /opt/jboss/keycloak/themes/keycloak
/welcome
--->Using cache
--->43fa6ae03ed0
Successfully built 43fa6ae03ed0
Successfully tagged idp_keycloak:latest
Starting idp_keycloak_1 ...  done
Recreating idp_https-portal_1 ...  done
```

3. システムが起動したことを確認
   コマンド:

```
$ docker ps
```

   出力:

```
 daiya@sandboxidp-portal: /idp$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
5be2742d0f25 steveltn/https-portal:1 "/init" 7 seconds ago
Up Less than a second 0.0.0.0:80->80/tcp, :::80->80/tcp,
0.0.0.0:443->443/tcp, :::443->443/tcp idp_https-portal_1 967acf6632e2
idp_keycloak "/opt/jboss/tools/do…" 13 minutes ago
```

```
Up 7 seconds 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp,
8443/tcp idp_keycloak_1
```

次に管理運用統合プレーンの構築方法と手順について述べる。なおこちらについても、前述した環境整備およびソースコードのダウンロード、展開については記述しない。構築時のディレクトリはダウンロードし展開したソースコードのトップディレクトリとする。手順と実際の操作については以下の通り。

1. 自身の組織に合わせて、ログの保存場所、利用しているソフトウェアのバージョンの環境変数をソースコードから確認し修正
2. 構成手順が記載された docker-compose.yml を起動
   コマンド:

```
$ docker-compose up -d --build
```

出力:

```
 daiya@albireo: /sandbox/logs$ docker-compose up -d --build
Creating network "logs_default" with the default driver
Building fluentd
Sending build context to Docker daemon 4.096kB
Step 1/6 :  ARG FLUENTD_VERSION
Step 2/6 :  FROM fluent/fluentd:$FLUENTD_VERSION
--->2c52d31e1e3d
Step 3/6 :  USER root
--->Using cache
--->91300b5c68f5
Step 4/6 :  ARG FLUENTD_PLUGION_VERSION
--->Using cache
--->a7b03f092253
Step 5/6 :  RUN gem install fluent-plugin-elasticsearch
--no-document --version
$FLUENTD_PLUGION_VERSION
--->Using cache
--->6b8cd305ecd2
Step 6/6 :  USER fluent
--->Using cache
--->c250897a63c7
Successfully built c250897a63c7
Successfully tagged logs_fluentd:latest
```

```
Creating logs_elasticsearch_1 ...  done
Creating logs_metricbeat_1 ...  done
Creating logs_fluentd_1 ...  done
Creating logs_kibana_1 ...  done
```

3. システムが起動したことを確認
   コマンド:
   `$ docker ps`
   出力:

```
daiya@albireo: /sandbox/logs$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
f3aa6ac2ca82 kibana:8.3.2 "/bin/tini -- /usr/l…" 10 seconds ago
Up 8 seconds 0.0.0.0:5601->5601/tcp, :::5601->5601/tcp logs_kibana_1
faf0266a9721 logs_fluentd "tini -- /bin/entryp…" 10 seconds ago
Up 8 seconds 5140/tcp, 0.0.0.0:24224->24224/tcp, 0.0.0.0:24224
->24224/udp, :::24224->24224/tcp, :::24224->24224/udp logs_fluentd_1
e301a2ba5302 docker.elastic.co/elasticsearch/elasticsearch:8.3.2
"/bin/tini -- /usr/l…" 10 seconds ago Up 9 seconds 0.0.0.0:9200
->9200/tcp, :::9200->9200/tcp, 9300/tcp logs_elasticsearch_1
07d40ec1f9ac docker.elastic.co/beats/metricbeat:8.3.2 "/usr/bin/tini
-- /u…" 10 seconds ago Up 9 seconds logs_metricbeat_1
```

これらように構築を行うために必要となるステップ数は、上記 1.、2. であり、確認を含んでも 3 ステップの非常に単純なコマンドで構築が可能になっており目的である再現性の高い認証認可基盤および管理運用統合プレーンのワンストップな構築の要件を満たしているといえる。

実行結果と確認

では、この構築が想定している構築結果となったかを確認する。まずコマンドの実行結果においては、上記のようにエラーはなく STATUS についても Up となっていることから正しく認証認可基盤および管理運用統合プレーンが構築されたことが示される。構築までかかる時間は必要なソフトウェアのダウンロードを行う際のネットワークの帯域にもよるが、認証認可基盤が約 3 分、管理運用統合プレーンが約 2 分で展開される。

次に実際にそれぞれのサービスの動作確認をブラウザからアクセスすることで行う。確認する項目は図 6.1 に示す (1)、(2)、(4) である。実際に確認した結果としては図 6.2、6.3、6.4 に示す通り、各システムが動作していることが確認できる。

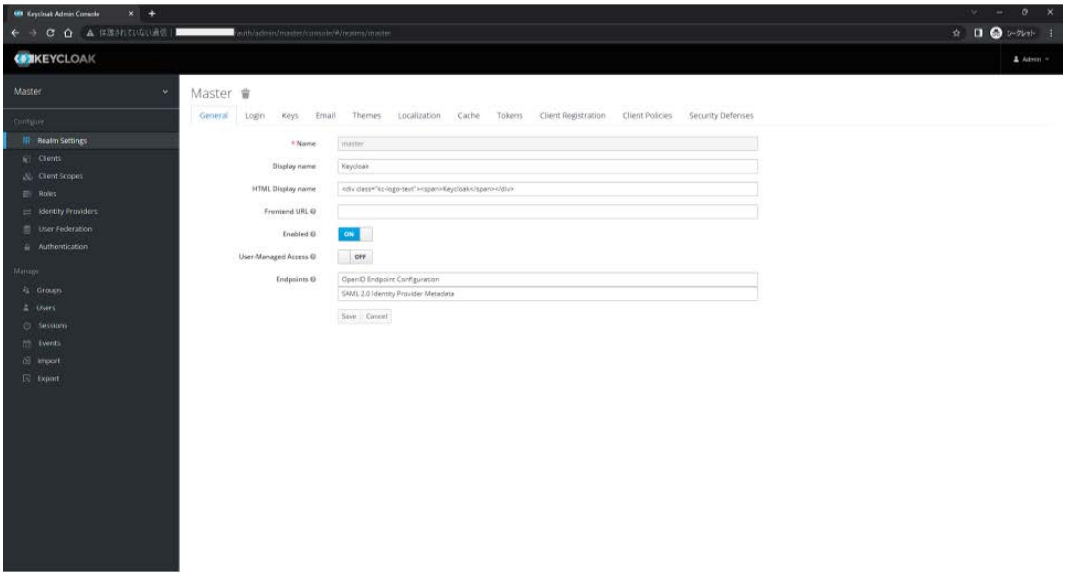

図 6.2　(1) 認証認可基盤の動作確認

発生した問題と解決策

プロトタイプ 1 では、実際にワンストップで認証認可基盤を構築することが示された。しかし継続的な運用の面で問題が発生した。問題となるのは、認証認可基盤構築時に出力されている “WARNING“である。これが意味するところは構築者が構築手順のステップ 1 を怠ったということである。構築者にインタビューを行ったところ、原因としてはソースコードに埋め込まれた環境変数のすべての項目を網羅的にチェックしていなかったことに起因していた。この問題が招く継続的な運用の面で問題としては、例えば “WARNING: The DB_PASSWORD variable is not set. Defaulting to a blank string.“つまりデータベースのパスワードを指定しなかった場合、パスワードがデフォルトのパスワードや空のパスワードとなり、

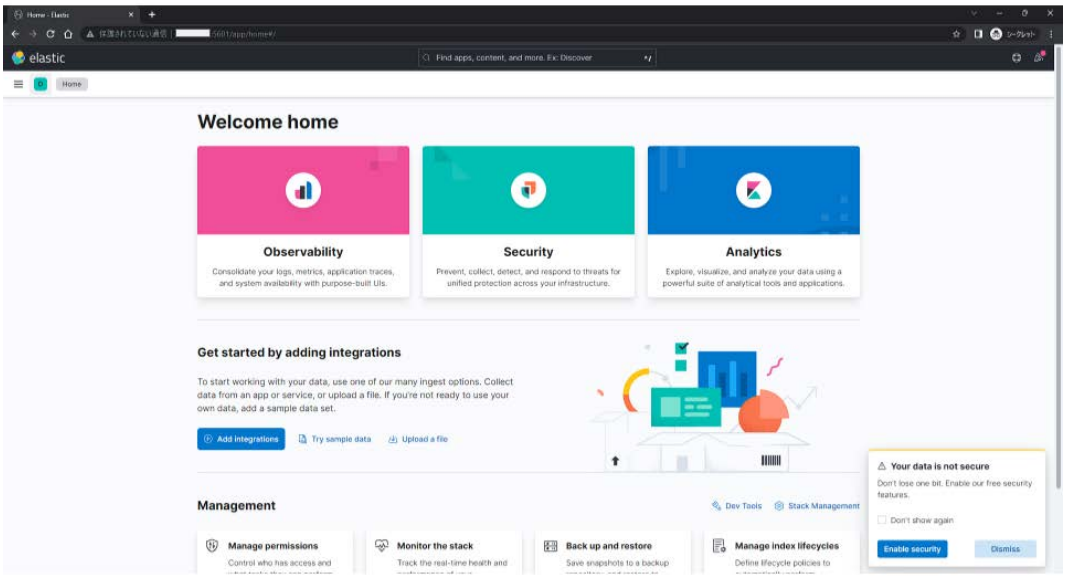

図 6.3 (2) ログ基盤の動作確認

システム運用の面で脆弱な環境を生み出す恐れがあり継続的な管理運用を行うことができない。また本システムに対して直接環境変数を修正した状態でオープンなコードリポジトリに保存した場合、クレデンシャルや組織の秘密の情報の漏洩が漏洩する恐れがある [114]。

そこで運用上の問題が発生し得る要素に対して必須の環境変数として定義し、これが満たされない場合においては構築されないようにすることによりこの問題の解決を図る。これをプロトタイプ 2 として実装し再度テストを行う。

6.2.6 プロトタイプ 2: 環境変数を網羅した構築

プロトタイプ 1 で発生した課題に対して解決策を講じたプロトタイプ 2 として、認証認可基盤および管理運用統合プレーンの構築と評価を行う。

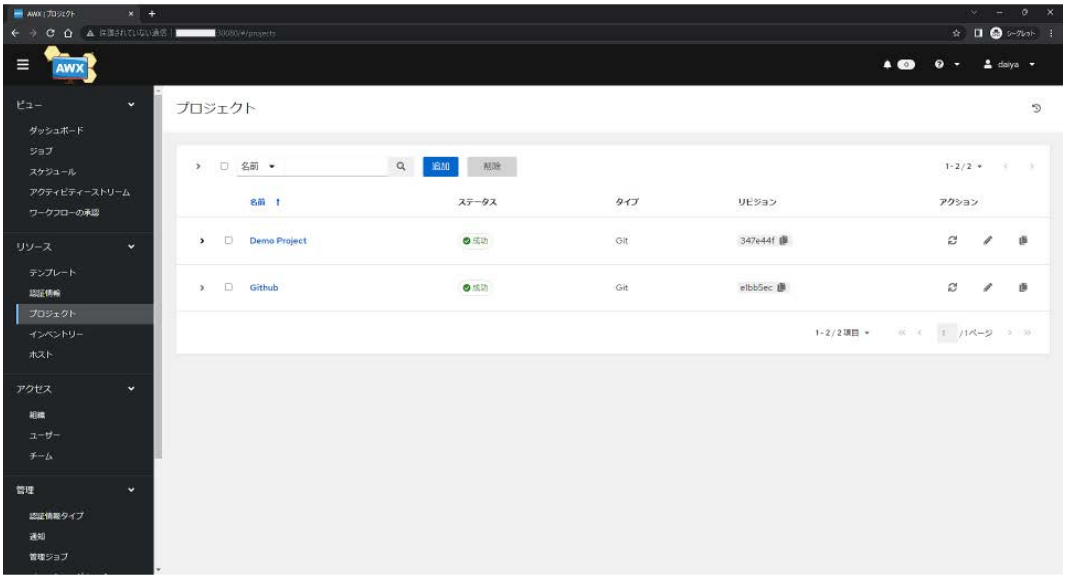

図 6.4 (3) 管理運用統合プレーンの動作確認

構築方法と手順

まず認証認可基盤の構築方法と手順について述べる。なお前述した環境整備およびソースコードのダウンロード、展開については記述しない。構築時のディレクトリはダウンロードし展開したソースコードのトップディレクトリとする。手順と実際の操作については以下の通り。

1. 自身の組織に合わせて、必須となっている初期ユーザー、パスワード、データベースのパスワード、証明書のドメイン名などの環境変数ファイルを作成

2. 自身の組織に合わせて、その他必要な環境変数を確認しソースコードから確認し修正

3. 構成手順が記載された `docker-compose.yml` を起動
   コマンド:
   ```
   $ docker-compose up --env-file idp_.env -d --build
   ```
   出力:
   ```
   daiya@sandboxidp-portal: /idp$ docker-compose up --env-file idp_.env
   -d --build
   ```

```
Building keycloak
Sending build context to Docker daemon 9.728kB
Step 1/2 :  FROM jboss/keycloak
--->011a708d97bf
Step 2/2 :  ADD themes/welcome /opt/jboss/keycloak/themes/keycloak/welcome
--->Using cache
--->43fa6ae03ed0
Successfully built 43fa6ae03ed0
Successfully tagged idp_keycloak:latest
Starting idp_keycloak_1 ...  done
Recreating idp_https-portal_1 ...  done
```

4. システムが起動したことを確認
   コマンド:
   ```
   $ docker ps
   ```
   出力:
   ```
   daiya@sandboxidp-portal: /idp$ docker ps
   CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
   5be2742d0f25 steveltn/https-portal:1 "/init" 12 minutes ago
   Up Less than a second 0.0.0.0:80->80/tcp, :::80->80/tcp, 0.0.0.0:443
   ->443/tcp, :::443->443/tcp idp_https-portal_1
   967acf6632e2 idp_keycloak "/opt/jboss/tools/do…" 32 minutes ago
   Up 7 seconds 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp, 8443/tcp
   idp_keycloak_1
   ```

次に管理運用統合プレーンの構築方法と手順について述べる。なおこちらについても、前述した環境整備およびソースコードのダウンロード、展開については記述しない。構築時のディレクトリはダウンロードし展開したソースコードのトップディレクトリとする。手順と実際の操作については以下の通り。

1. 自身の組織に合わせて、必須となっているログの保存場所、利用しているソフトウェアのバージョンの環境変数ファイルを作成
2. 自身の組織に合わせて、その他必要な環境変数を確認しソースコードから確認し修正
3. 構成手順が記載された `docker-compose.yml` を起動
   コマンド:

```
$ docker-compose up --env-file logs_.env -d --build
```

出力:

```
 daiya@albireo: /sandbox/logs$ docker-compose up --env-file logs_.env -d
--build
Creating network "logs_default" with the default driver
Building fluentd
Sending build context to Docker daemon 4.096kB
Step 1/6 :  ARG FLUENTD_VERSION
Step 2/6 :  FROM fluent/fluentd:$FLUENTD_VERSION
--->2c52d31e1e3d
Step 3/6 :  USER root
--->Using cache
--->91300b5c68f5
Step 4/6 :  ARG FLUENTD_PLUGION_VERSION
--->Using cache
--->a7b03f092253
Step 5/6 :  RUN gem install fluent-plugin-elasticsearch --no-document
--version
$FLUENTD_PLUGION_VERSION
--->Using cache
--->6b8cd305ecd2
Step 6/6 :  USER fluent
--->Using cache
--->c250897a63c7
Successfully built c250897a63c7
Successfully tagged logs_fluentd:latest
Creating logs_elasticsearch_1 ...  done
Creating logs_metricbeat_1 ...  done
Creating logs_fluentd_1 ...  done
Creating logs_kibana_1 ...  done
```

4. システムが起動したことを確認

コマンド:

```
$ docker ps
```

出力:

```
 daiya@albireo: /sandbox/logs$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
f3aa6ac2ca82 kibana:8.3.2 "/bin/tini -- /usr/l…" 8 seconds ago
Up 5 seconds 0.0.0.0:5601->5601/tcp, :::5601->5601/tcp logs_kibana_1
faf0266a9721 logs_fluentd "tini -- /bin/entryp…" 8 seconds ago
```

```
Up 5 seconds 5140/tcp, 0.0.0.0:24224->24224/tcp, 0.0.0.0:24224
->24224/udp, :::24224->24224/tcp, :::24224->24224/udp logs_fluentd_1
e301a2ba5302 docker.elastic.co/elasticsearch/elasticsearch:8.3.2
"/bin/tini -- /usr/l…" 10 seconds ago Up 8 seconds 0.0.0.0:9200
->9200/tcp, :::9200->9200/tcp, 9300/tcp logs_elasticsearch_1
07d40ec1f9ac docker.elastic.co/beats/metricbeat:8.3.2 "/usr/bin/tini
-- /u…" 10 seconds ago Up 9 seconds logs_metricbeat_1
```

プロトタイプ 1 と比較すると構築を行うために必要となるステップ数は、上記 1.、2.、3. と増加し、確認を含めると 4 ステップとなったものの非常に単純なコマンドで構築が可能になっており目的である再現性の高い認証認可基盤および管理運用統合プレーンのワンストップな構築の要件は引き続き満たしているといえる。また問題であった各環境に必要となる環境変数を最低限網羅することについても解決を図った。

実行結果と確認

プロトタイプ 1 と同様の結果を得られたため省略する。

6.2.7 評価結果

以上で評価結果としては、構築の経験およびノウハウがなくとも基本的なコマンド操作を行うことが可能であれば、認証認可基盤、管理運用統合プレーンを構築することが可能であることが示された。しかし構築をワンストップにするあまり、必要となる環境変数等の諸設定を見落とす問題が発生した。そのため最終的に手順事態は増えてしまうが、諸設定を行うステップを追加することで継続的な管理運用を行うことが可能となる環境構築を実現した。これにより 4.1 節にある、認証認可基盤全体のフレームワーク化の継続的な管理運用が容易なシステムアーキテクチャの要件を満たすと考える。

6.2.8 さまざまな環境への対応

次に認証認可基盤や管理運用統合プレーンが再現性をもって様々な環境で構築することが可能であるかを検証する。本研究のターゲットとなる環境で一般的に用いられるOSで構築が可能であることを検証する。今回の環境は、Ubuntu 22.04 Server、Windows 10の環境を用いて検証を行う。また対象外ではあるが、現在の潮流としてクラウドインフラ上でも同様に構築が可能であるか検証を行う。

ケース1:Ubuntu 22.04 Server、Windows 10

本研究の対象外ではあるが、ハードウェア要件としてARMアーキテクチャ[115]のCPUを利用した環境での構築は非推奨とする。これは利用しているdockerのコンテナ化が一部利用できない可能性があるためである。そのため、Intelアーキテクチャの CPUを利用することが前提となる。

まずUbuntu 22.04 Serverでの構築について検証する。これについてはすでに、6.2.3項、6.2.5項、6.2.6項で示した通り、要件通りに認証認可基盤および管理運用統合プレーンを構築することが可能であった。

次にWindowsでの環境について示す。Windows環境を用いる想定としては、動作確認として開発環境で認証認可基盤を試すことを想定している。現在のシステム運用の潮流を考えた場合、Windows環境にて認証認可基盤を実践的に運用することは考えにくい。そのためWindows 10での構築について検証する。6.2.2項でも示したが、前提条件としてdockerが動作する環境がWindowsにおいても必要となっている。今回の検証ではDocker [70]が提供しているDocker Desktop [116]を用いることとする。

構築結果としては図6.5、図6.6に示す通り、Docker Desktop上に認証認可基盤および管理運用統合プレーンが構築されていることがわかる。

ケース2:AWS、GCP

本研究では対象外ではあるが、クラウドインフラ上での構築可能性について述べる。今回は、一般的に用いられているAWS、GCP上での構築について述べる。

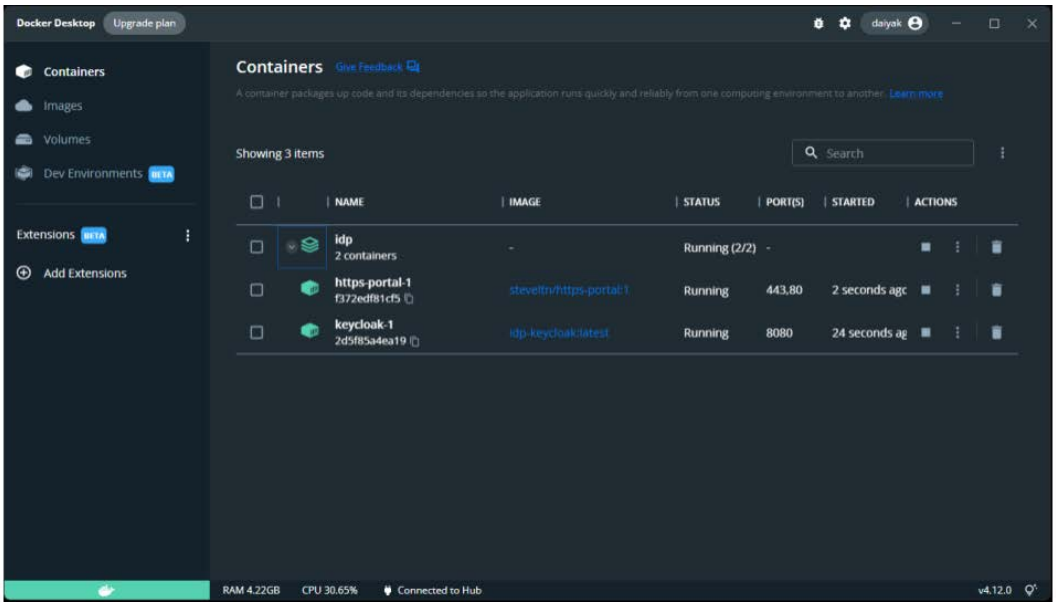

図 6.5 Windows10 Docker Desktop 上での認証認可基盤の構築

双方の大まかな構築のアーキテクチャを図 6.7 に示す。

AWS で docker を利用した構築を行う場合については、Amazon ES2 で 6.2.3 項と同様の Linux 環境を構築して認証認可基盤および管理運用統合プレーンを直接構築するか、事前にローカル環境で作成した認証認可基盤および管理運用統合プレーンの dokcer image を Amazon ECR 上に登録し、Amazon ECS 上に登録した docker image をもとに docker コンテナを展開することで構築が可能である。

GCP で docker を利用する場合についてもアーキテクチャとしてはどうようであり、GCE で 6.2.3 項と同様の Linux 環境を構築して認証認可基盤および管理運用統合プレーンを直接構築するか、事前にローカル環境で作成した認証認可基盤および管理運用統合プレーンの dokcer image を Container Registry 上に登録し、Cloud Run 上に登録した docker image をもとに docker コンテナを展開することで構築が可能である。

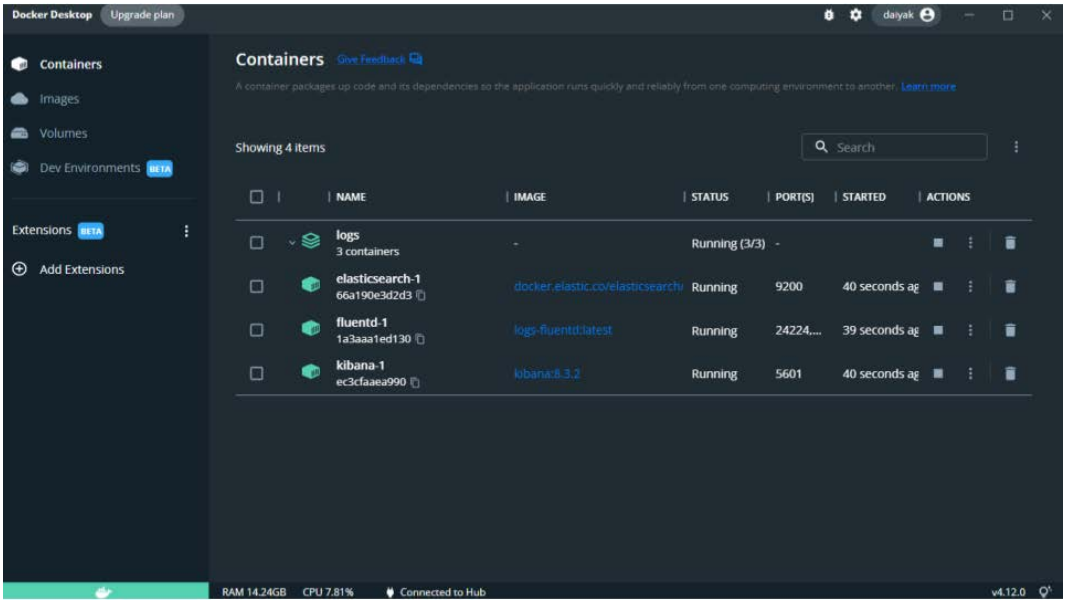

図 6.6 Windows10 Docker Desktop 上でのログ基盤の構築

6.2.9 評価結果

以上で評価結果としては、様々な環境において認証認可基盤や管理運用統合プレーンを再現性をもって構築することが可能であることを示した。これにより広く一般的に利用されている OS ディストリビューションおよびクラウドサービスで利用可能となり、4.1 節にある、認証認可基盤全体のフレームワーク化の環境への依存が少ないシステム構成、再現性の高い認証認可基盤のワンストップな構築の要件を満たすと考える。

6.2.10 まとめ

以上の 6.2.9 節、6.2.9 節の評価結果から、4.1 節で提示した本研究の目的の一つである認証認可基盤全体のフレームワーク化を満たすことを示す。

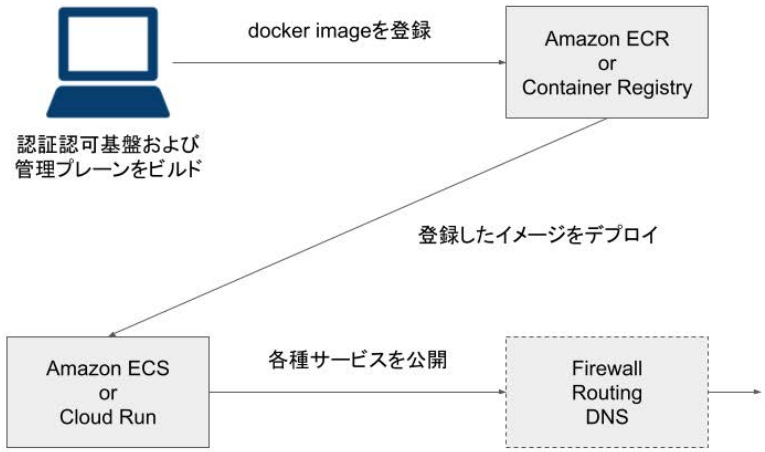

図 6.7　AWS、GCP への構築

6.3. 管理運用統合プレーンの評価環境について

管理運用統合プレーンの評価を行う環境について示す。今回の環境としては、共同研究を行っているプロジェクト内での環境と KMD 内の Network Media プロジェクトでの環境を取り扱う。

6.3.1 共同研究内における活用

共同研究のプロジェクト内で管理運用をしている認証認可基盤について述べる。

目的

共同研究内におけるセキュリティインシデントレスポンス基盤を複数組織で利用するための認証認可基盤として活用している。認証認可基盤の主な利用方法としては、インシデントを組織間で共有を行うファイルサーバーへのアクセス制御、

インシデント脅威情報を共有するトラブルチケットシステムへのアクセス制御を行うことを目的としている。

参加人数

ユーザーが 10 名、うち管理者が 2 名、内訳としては共同研究先が 6 名、慶應義塾大学が 4 名となっている。

アーキテクチャ

全体のアーキテクチャについて図 6.8 に示す。認証認可基盤にはサービスが複

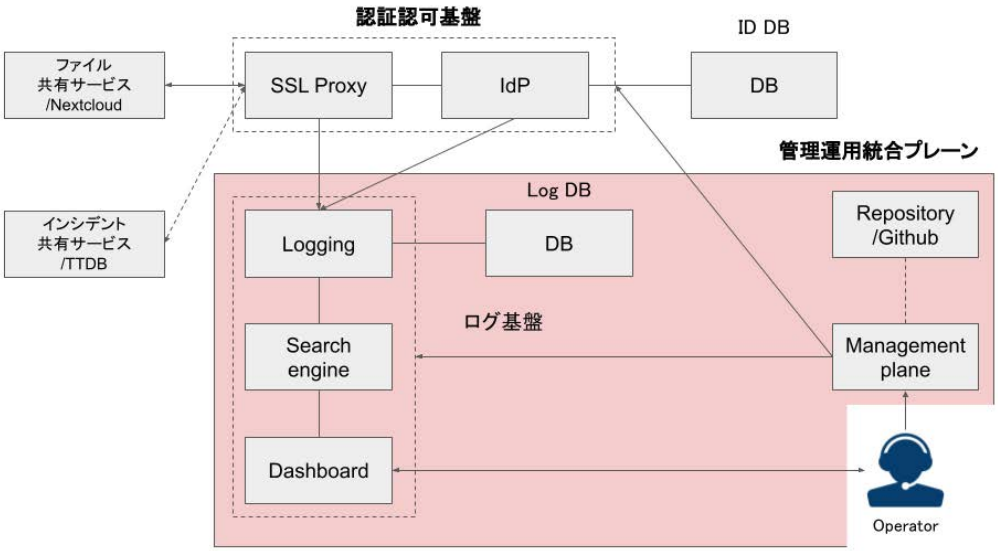

図 6.8　共同研究内でのアーキテクチャ

数接続されており、Digital Identity および各種ログは外部の DB を利用している。認証認可基盤およびログ基盤は管理運用統合プレーンで管理することを想定している。運用テンプレートは外部サービスの Github に保存しているもの呼び出すことで利用している。

6.3.2 KMD 内における活用

KMD の Network Media プロジェクトで管理運用をしている認証認可基盤について述べる。

目的

KMD における研究科ホームページや elearning システムなどの各種サービスの認証認可機能を認証認可基盤に委任するためのテストケースとして Network Media 内で運用を行っている。OIDC を用いた認証連携や SSO の検証、FIDO などの複数の認証要素の検証を行うことを目的としている。

参加人数

Network Media 内のユーザーが 6 名、うち管理者が 1 名となっている。

アーキテクチャ

全体のアーキテクチャについて図 6.9 に示す。

認証認可基盤にはファイル共有サービスが接続されており、Digital Identity および各種ログは各マシン内に DB を作成し利用している。認証認可基盤およびログ基盤は管理運用統合プレーンで管理することを想定している。運用テンプレートは外部サービスの Github に保存しているもの呼び出すことで利用している。

6.4. 管理運用統合プレーンの運用事例と実証と評価

6.4.1 共同研究内における運用事例

共同研究のプロジェクト内で管理運用をしている認証認可基盤での運用事例を述べる。

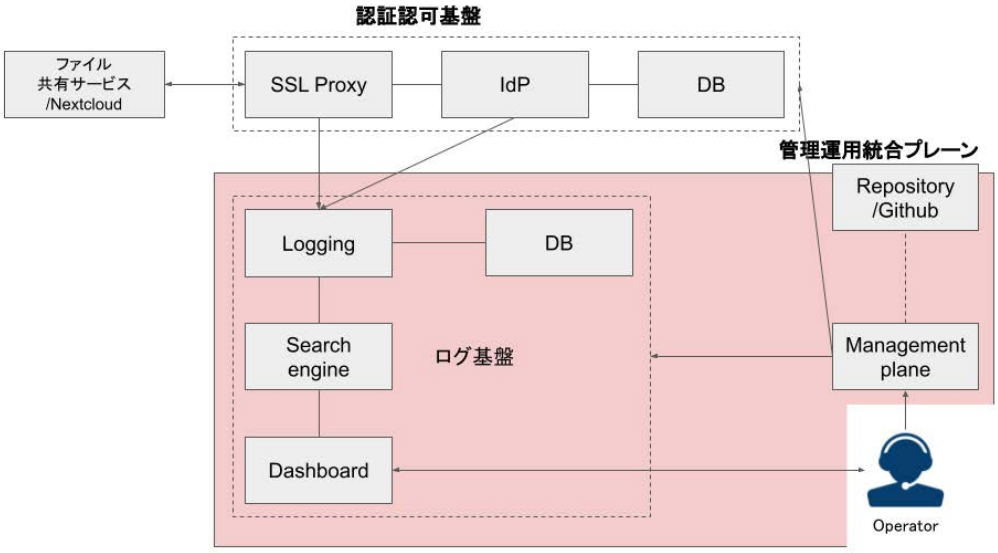

図 6.9　KMD でのアーキテクチャ

6.4.2　ケース 1: アカウントの新規追加

利用するサービスに対して新規にアカウントを追加する場合の運用について述べる。

発生した事象

共同研究先および慶應義塾大学側でサービスを利用する新規ユーザーを登録する必要がある。本サービスはアカウントを有している人のみが利用可能となるサービスを想定しているため、ユーザー個々人で新規にアカウントを作成できない仕組みにしている。そのため、管理者側で新規ユーザーを作成する必要が発生した。

実際に行った運用

管理運用統合プレーンから、ユーザー新規作成のテンプレートを開きユーザー登録に必要となるパラメーターを確認する。次に、ユーザーに対して必要となる

パラメーターの情報を収集し環境変数としてテンプレートに入力する。最後にテンプレートを実行し実行結果を確認を行う。実際のテンプレート実行の様子を図 6.10 に示す。

図 6.10　アカウント新規作成のテンプレート実行

これについて従来の作業としては、まず IdP にアクセスしユーザー一覧のページを参照する。次にユーザー登録に必要となるパラメーターを確認し、ユーザーに対して必要となるパラメーターの情報を収集する。その後、取得した情報をもとにユーザー名が利用されていないことを事前に確認し、存在していないことを確認したのちに、アカウント作成の画面にアクセスし、必要な項目を手作業で入力していく。最後にアカウントが作成されたことをユーザー一覧から検索することで確認する。という流れとなっている。

運用に対するコメント

本運用に対する評価について管理者からの定性評価を示す。本運用を行った管理者は日常的に認証認可基盤を運用していないが、過去に本研究以外の認証認可

基盤に対してユーザー追加を行った経験がある人物である。

本運用に対しての意見としては、“今回のように数人のユーザーが対象である場合については管理運用統合プレーンを使わずとも既存の手法通りに IdP の GUI を操作しても運用コストはあまり変わらないように感じる “、“しかし、ドキュメントを熟読しなければそもそも IdP のどの部分を操作しなければならないかを判断できないため、運用経験がない者に対しては迷わずにアカウントを作成できるし、他の運用者に指示を行う手間が省ける “という評価を得た。また、“これが大量のユーザーを対象としなければならない場合、操作というコストが大きく削減できテンプレートを実行すれば、待つだけで良いので楽 “というコメントを得た。

6.4.3 ケース 2: アカウントの停止および削除

利用するサービスに対して既存のアカウントの停止および削除する場合の運用について述べる。

発生した事象

共同研究先のアカウントにサービスを検証する際に用いていた管理者権限を持つテストユーザーを発見し、すぐさま停止しアカウントを削除する必要が発生した。この際にアクセスログを確認しアカウントが悪用されていないかについても調査を行った。

実際に行った運用

管理運用統合プレーンから、ユーザー削除に必要となるパラメーターを確認する。次に必要となるパラメーターの情報を環境変数としてテンプレートに入力する。最後にテンプレートを実行し実行結果を確認を行う。実際のテンプレート実行の様子を図 6.11 に示す。またアクセスログの確認については、管理運用統合プレーンのログダッシュボードにアクセスする。次にダッシュボードの検索窓から調査したい項目 (今回はユーザー名) を指定し、調査する内容を入力し検索する。

検索結果としてダッシュボードに表示された各種項目を確認する。という流れになっている。実際に確認したアクセスログを図 6.12 に示す。

図 6.11　アカウントの停止および削除のテンプレート実行

これについて従来の作業としては、まず IdP にアクセスしユーザー一覧のページを参照する。次に対象となるユーザー名を検索し削除を行う。最後にアカウントが削除され存在しないことをユーザー名の検索によって確認を行う。という流れとなっている。アクセスログの確認としては、認証認可基盤が構築されているマシンに ssh でアクセスし、Proxy および IdP のログをログが保存されているディレクトに移動し検索する。この時の検索はログ内に特定のユーザー名が含まれるものを絞りこみを行う方式をとる。出てきたすべてのログを目視ですべて確認し不審なアクセスがないことを確認する。という流れになっている。この時実際に確認するログの例を図 6.13 に示す。

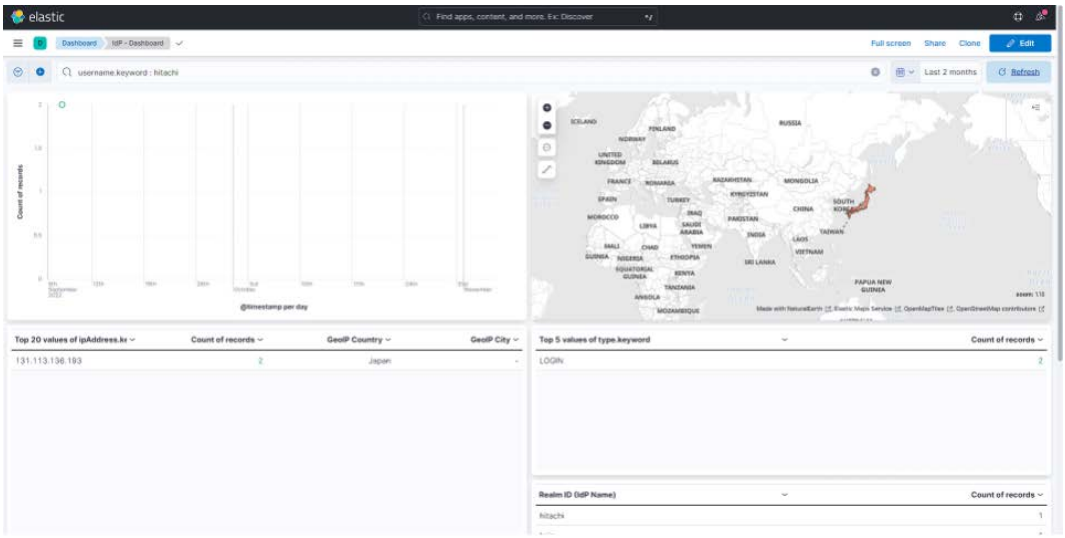

図 6.12　既存アカウントが利用されていないかの確認

運用に対するコメント

本運用に対する評価について管理者からの定性評価を示す。ユーザー作成時と同様の人物で本運用を行った管理者は日常的に認証認可基盤を運用していないが、過去に本研究以外の認証認可基盤に対してユーザー削除を行った経験がある人物である。

本運用に対しての意見としては、“ユーザー作成時と同様に、大量のユーザーを対象とした場合は効力が得られると感じるが、数人のユーザーが対象である場合については管理運用統合プレーンを使わずとも既存の手法通りに IdP の GUI を操作しても運用コストはあまり変わらないように感じるが、同様に運用経験がない者に対しては有効であると思う “という評価を得た。また、“大量にユーザーを削除しなければならない状況は、年度末の人事異動などの特定のイベントの時なのでイベント毎にテンプレートを作成できないか “というコメントを得た。

直近2日で605行分のログ（wc -l access.log 605 access.log）

図 6.13 確認しなければならないログの例

6.4.4 ケース 3: ログ基盤停止における状況確認および再起動

ログ基盤にアクセスができなくなり、状況の確認およびログ基盤の再起動を行う場合の運用について述べる。

発生した事象

認証認可基盤のアクセスログを確認するために、ログ基盤にアクセスしたところサービスが動作していなかった。ログ基盤動作していない原因を調査しログ基盤の再起動を行った。

実際に行った運用

管理運用統合プレーンから、ログ基盤が構築されているマシンに対して死活管理のテンプレートを実行。マシン自体は起動していることがわかったため、ssh を用いて対象となるマシンにアクセスし、docker の動作状況の確認。ダッシュボー

ドがクラッシュし停止していたため、その後管理運用統合プレーンからログ基盤を再起動のテンプレートを実行した。数分後ログ基盤にアクセスし正常に動いていることを確認した。という流れになっている。

これについての従来の作業については、ログ基盤が構築されているマシンに ssh を用いてアクセス、ログ基盤として活用しているソフトウェアが起動していることをすべて確認。該当するソフトウェアを特定したのちに停止原因を当該ログが保存されているディレクトリにアクセスしログファイルを調査する。ダッシュボードがクラッシュし停止していたため、当該ソフトウェアをソフトウェア指定のコマンドで再起動する。数分後ログ基盤にアクセスし正常に動いていることを確認した。という流れになっている。

運用に対するコメント

本運用に対する評価について管理者からの定性評価を示す。ユーザー作成時と同様の人物で本運用を行った管理者は日常的に業務としてログを管理、調査を行っている人物である。

本運用に対するコメントとしては、“ターゲットとしているマシンに直接システムを構築しているものに比べて調査する箇所が少なく原因の特定がしやすいと思う。またソフトウェアを起動する際にソフトウェア独自の方法をいちいち覚えておかなくて済む“という評価を得た。しかし課題として、“今回のようにログ基盤がダウンしている場合、原因の特定のためにログ基盤を確認することができないため管理運用統合プレーン側からログは取得できなくともシステムの動作情報を取得できないか“という運用テンプレートの追加に関する意見を得た。

6.4.5 共同研究内における評価まとめ

以上から共同研究内の環境における管理運用から得られた評価についてまとめる。本研究における管理運用統合プレーンの果たすべき役割は、4.4 節でも示した通り、認証認可基盤の管理運用の簡略化である。

まず全体の所感としては、管理運用統合プレーンを用いることで、従来のシステムの管理運用方法と比べて認証認可基盤の管理運用の簡略化を行えていると考えられる。具体的には、複数マシンへのアクセスやシステムの稼働状況を収集するための繰り返し行わなければならない基本的な操作を管理運用統合プレーンに登録されている運用テンプレートを用いることでによって削減することができているためである。また、運用テンプレートによる繰り返し行わなければならない基本的な操作の簡略化という点では、アカウント作成、削除のアカウントマネジメントにも反映できている。これは 5.3.1 項、5.3.2 項、5.3.3 項で設計した内容が想定通り機能している。

またユーザーアクセスとアカウントの状況把握については 5.3.1 項で設計した想定の通り、該当するアカウントを図 6.12 に示すように検索し、アクセス回数やアクセス元、ログイン状況から判断をするという行動を取っている。

しかし課題としては、本研究では認証認可基盤の管理運用をサポートするために管理運用統合プレーンを設計しており、管理運用統合プレーンの管理運用をサポートするための管理運用統合プレーンを設計していない。そのためケース 3 で発生した管理運用統合プレーン内のログ基盤に問題が発生した場合に利用可能となるテンプレートを用意していない。例えば今回のような場合が発生した場合、従来の運用通り当該マシンにアクセスし運用を行う必要がある。しかし従来の方法比較すると 5.2.1 項で設計した通りモジュール化をしているためシステムの状態の把握、該当するソフトウェアの再起動などの操作の簡略化を行っている。これは “ソフトウェア独自の方法をいちいち覚えておかなくて済む “というコメントからも確認できている。

とはいえ根本的な問題は解決できておらず、現状としては、本研究の運用テンプレートは作成方法を学習することで自由に追加できるように設計しているため、個々人で必要となるテンプレートを用意することを提示している。しかしコメントの通り、“管理運用統合プレーン側からログは取得できなくともシステムの動作情報を取得できないか “というような要望は少なからず出てくるであろう。そのため本研究としては取り扱わないが、今後の課題としてどこまでを運用テンプレートとして用意しサポートするのか、また個々人で作成した運用テンプレートを共

有する仕組みづくりを行う必要があるのかという検討を行う予定である。

6.4.6 KMD 内における運用事例

KMD Network Media 内で管理運用をしている認証認可基盤での運用事例を述べる。本運用事例は実践での運用を想定したテストであり筆者が運用を行った。

6.4.7 ケース 4: 特定アカウントに対する攻撃を想定したテスト

特定アカウントに対する大量のアクセスによる攻撃を想定し、アカウントの状況の確認およびアカウントの停止について述べる。

発生した事象

短時間で特定アカウントに対して大量のログインエラーが発生していることをログ基盤で確認し、該当するアカウントに対する操作を決定する。

実際に行った運用

管理運用統合プレーンのログダッシュボードにアクセスし、短期間に特定のログイン処理が行われていることを確認。ダッシュボードから該当するアカウント特定し、検索窓からアカウントを絞り込むことでログイン処理の結果を確認。特定の IP アドレスから大量のログインエラーは確認できたもののログインが成功していないことが判明し、アカウントへの特定の操作は一時見送った。実際に確認したアクセスログを図 6.14 に示す。

6.4.8 ケース 5:IdP のシステムアップデート

利用している認証認可基盤のバージョンのアップデートを行う際のシステムの更新および再起動について述べる。

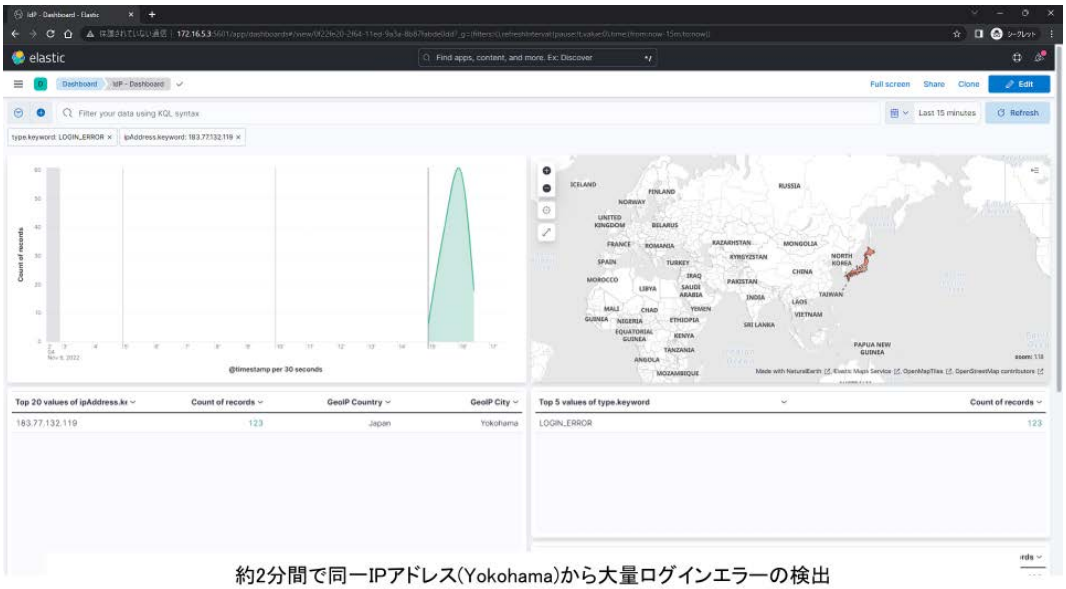

図 6.14　大量のアクセスとログインエラーの確認

発生した事象

認証認可基盤の IdP にバグが見つかり、利用しているバージョンからアップデートを行う必要がある。IdP のアップデート後にサービスが再び起動することを確認する。

実際に行った運用

管理運用統合プレーンから認証認可基盤を停止するテンプレートを実行する。停止後、管理運用統合プレーンのリビルドを行うテンプレートに指定のバージョンを環境変数として指定し実行する。実行後自動でリビルドしたコンテナが起動するので、ブラウザから認証認可基盤にアクセスし動作を確認する。

6.4.9　KMD 内における実証まとめ

以上から KMD 内の環境における管理運用から得られた所感についてまとめる。

ケース 4 で行った特定アカウントに対する攻撃を想定したテストにおける不審なアクセスログの調査については考察する。従来方法では膨大なアクセスログを単位時間での送信先の IP アドレス数をカウントし攻撃元の IP アドレスを特定、その後どの地域からのログインなのかを調査する。調査結果を IdP にアクセスしログと照らし合わせどのサービスのどのアカウントに対して攻撃が行われているのかを特定するという一連の運用となる。この運用をログ基盤のダッシュボードを用いることで一括して可視化し、単位時間で絞り込むことで対象を調査することが可能となりログ調査の一点の操作を簡略化している。ケース 5 では、認証認可基盤を構築している全システムの停止、再起動を行う運用である。従来方法では各システムにそれぞれアクセスしソフトウェア独自のコマンドで停止および再起動を行う必要がある。これを管理運用統合プレーンの停止、起動テンプレートを実行することで認証認可基盤全体のシステムを一括で操作できることを確認した。

6.5. 管理運用統合プレーンの実証と評価まとめ

6.4.1 項、6.4.6 項より、管理運用統合プレーンの実証と評価を 5.4 項の目的と照らし合わせてまとめる。

まず認証認可基盤の状況把握とユーザーアクセスとアカウントの状況把握については、確認したケースにおいて 4.8 節で示した必要となるデータが要件通りログ基盤に収集および可視化され、被験者が状況把握のために活用できていることを確認した。運用においても 4.8 節で設計したデータをログダッシュボードを活用することで状況の把握に活用することにより、従来のソフトウェア毎に分散していたログ集約することでログを探索するという負担を軽減し、ログ情報の中から必要な情報を見つけ出し利用可能な状態に成型するという負担についても軽減している。次に被験者の評価としては、今回の 5 つのケースの事象において本研究で選定し定義したダッシュボードの各種データの項目として不足している、どこを見れば良いのかわからず運用ができないという意見は得られなかったが、これは被験者の経験に左右されるところが大きく、状況把握のための検索キーや注視するべき点をどの項目にするのかという判断基準がノウハウに依存すると感じ

た。本研究はあくまでも運用の簡略化を目指すものであり、能力やノウハウが全くない運用者を対象とはしていない。しかし最低限必要となる知識や技能についても今後サポートできる仕組みを導入することでより対象をより広くとることができると考える。

認証認可基盤の管理運用の簡略化については、認証認可基盤内のシステムを管理するケースにおいてテンプレートが活用され、モジュール化によりシステムの状態の把握や操作が簡略化されていると考察する。ケース 3 で見られた停止している機能およびソフトウェア状況の確認については、それぞれの機能に実際に ssh を用いてアクセスし稼働状況の確認およびログの探索および該当するエラー箇所の確認を行うという従来方法に比べ、ログ基盤のダッシュボードを活用することで一括して状況を確認することが可能となっている。しかし同様にケース 3 で見られた管理運用統合プレーンのための管理運用統合プレーンがないという問題に対しては、今後どこまで本研究のアーキテクチャでサポートする必要であるのか見当を行う必要があり未解決の課題である。

アカウントマネジメントの簡略化については、認証認可基盤の管理運用の簡略化と同様に、5.3.2 項、5.3.3 項で設計した運用テンプレートが想定通り活用されており、コメントとしてもポジティブな印象を確認できた。従来方法と比較するとケース 1、2、4 については IdP の管理画面へのログイン、アカウント操作に対する GUI の単調な繰り返し操作をテンプレートを活用することによってアカウント操作に必要な情報の入力とテンプレートの起動という少ない工程で実施できることを確認した。しかしこの項目についても例外的な稀なケースが発生した場合、そのための想定した運用テンプレートを用意するべきかという点については課題となっている。しかし現段階においても用意した運用テンプレート以外に個々人で作成が可能であるため、実践で利用していくことでより組織に合った管理運用統合プレーンに成長することが期待できる。

以上で、4.1 節で提示した本研究の目的の一つである認証認可基盤の管理運用の簡略化を満たすことを示す。

6.6. 管理運用統合プレーン上の運用コードの活用状況

最後に管理運用統合プレーンで用いる運用コードとテンプレートの活用状況について述べる。現在実装されているテンプレートは、5.3.2 項にある 12 種類のテンプレートである。今回の 5 つの例ではそのうちの 8 種類が利用された。

利用されなかったテンプレートについて考察する。まずコンテナを再起動する“container-restart“である。これは運用においてシステムが正常に動いていない場合、一度停止し、状況を確認してから起動するという運用を行うことが多く、原因を把握せずにとりあえず再起動を行うというシチュエーションが少ないことが原因であると考察する。次に指定したアカウントのユーザー情報を取得する“get_user_info“である。これはログ基盤において確認が可能であるための利用されることがなかった。しかし付随している他のテンプレートと依存関係にあるための必要となっている。次に、指定したアカウントの指定したユーザー情報を更新 “update_user_info“である。これについてはケース 1 のコメントにもある通り年度初めや期間の区切りという特定の時期に利用されることが想定され、今回のケースではそのタイミングがなく利用されなかったと考察する。最後に指定したアカウントの指定したパスワードに変更する “update_user_password“である。これについては組織内でパスワードを誰が操作することが可能であるのかというポリシーの違いが起因しており、今回の 2 つの事例においては管理者がユーザーのアカウントのパスワードを変更してはならず、個々人でパスワードを設定するというポリシーを取ったため利用しなかった。

これらのことから用意したテンプレートの中で “container-restart“については必要がない可能性が高く修正する必要があると考える。またテンプレートの不足については、ケース 3 や 6.5 項でも述べたが今後も検討の余地があるが、当然すべてを網羅することはできないため必要になったときに自由にテンプレートを作成可能であるということで要件を満たす。

6.7. 本提案のアーキテクチャの評価

本研究で提案した認証認可基盤および管理運用統合プレーンアーキテクチャについての評価を述べる。

6.7.1 システムポリシーの適用

組織のシステムポリシーの適用について述べる。6.2 節で示した結果から従来方法と比較して、自組織の環境に影響を受けず、認証認可基盤および管理運用統合プレーンアーキテクチャを導入することが可能であることを示した。また付録 A を参照すること、従来方法のようにソフトウェア間の依存性を考慮することなく、組織のシステムポリシーに合わせてソフトウェアを選択可能となることについても示した。

6.7.2 認証認可基盤のシステム管理運用

従来の方式では、認証認可基盤と管理運用が別の分野としてアーキテクチャが分かれており、システムの状況把握を行うことが困難であり、また既存のアーキテクチャで管理運用を行う場合にはシステムポリシーを担保することができなった。本提案アーキテクチャでは、管理運用統合プレーンによりシステムの管理と運用を統合したアーキテクチャを実現することで、6.5 節で示した通り運用者が一元的に状況を把握でき、かつ状況に合わせて運用をシステムに対してテンプレートを用いることで簡潔に行うことが可能であることを示した。また認証認可基盤を構成しているソフトウェアが異なっていたとしても、付録 B で定義しているモジュール毎で同様の管理情報を得ることが可能となり同様の結果を得ることを確認した。

6.7.3 認証認可ポリシーの適用

組織の認可ポリシーの適用について述べる。従来のアーキテクチャでは利用している認証認可システムおよびアーキテクチャを原因とした特定の認証認可ポリ

シーしか対応できない、または自由に認証認可ポリシーを適用できなういという課題があった。本提案のアーキテクチャでは、システムポリシーが守られるアーキテクチャとなっており、認証認可ポリシーを適用可能な IdP(Identify Provider) を選択可能となっている。また 5.3.2 節で設計したテンプレートを活用することにより認証認可ポリシーを IdP(Identify Provider) に簡易的に適用することが可能となっており、6.5 節の評価と管理者からのコメントから認証認可ポリシーを一元的に適用可能なアーキテクチャとなっていることを示した。

6.7.4 テンプレートとポリシーの適用の拡張性

また 5.3.2 節でテンプレートは設計されているもの以外にも、付録 F 節で定義しているテンプレートを組織のポリシーに合わせて自由に作成できる拡張性を有している。そのため管理運用統合プレーンアーキテクチャに実装されていないシステムポリシー、認証認可ポリシーであっても同様の手法で各組織に合わせて活用可能である。

6.8. コミュニティへの貢献状況

本研究成果の各コミュニティへの貢献について述べる。

6.8.1 IOT 分野への貢献

本研究は、実践的な認証認可基盤を実現するための設計、要素の検討と実装について、情報処理学会の DICOMO シンポジウム (マルチメディア、分散、協調とモバイル) において論文執筆と研究発表を行っている。2018 年度はインターネットとサービス分野において “学内サービスにおける多要素認証の導入における検討 “、2019 年度はネットワークシステム分野において “学内サービスパスワードレス化の実現性の検討 “というタイトルで発表を行い様々な議論を行った。また、2018 年度においては、論文の有用性を認められヤングリサーチャ賞を受賞した。

これらの成果から本研究の IOT 分野における学術への影響は少なからずあり、貢献しているものと考える。詳細は巻末、業績リストに記載する。

管理運用統合プレーンの提案と実装、運用については、情報処理学会 IOT 研究会主催 IOTS 2021 において “研究科単位での学内サービスの ID 統一化に向けた IdP の導入および運用についての検討 ‘というタイトルで発表し議論を行った。また本発表を通して 6.8.2 節で述べる統合認証シンポジウムでの講演に繋がったことから本分野への社会貢献は少なからずあると考える。詳細は巻末、業績リストに記載する

6.8.2 統合認証シンポジウムへの貢献

第 14 回統合認証シンポジウムにおいて、“研究科単位での学内サービスの ID 統一化に向けた IdP の導入および運用についての検討 “というタイトルで主催者からの依頼を受け講演を行った。本シンポジウムの対象は、“統合認証に関心を持つ研究者、技術者、情報システム担当者、学生 “であり、目的は、“大学間認証連携「学認」に 260 以上の組織が参加し、190 万人以上の ID が連携するようになりました。これらにより、利用者の利便性の向上と管理コストの削減だけでなく、大学間連携、地域連携、さらに大学を超えた共同研究の可能性が拡大しています。新型コロナウィルスの影響で、教育・研究に関わる多くの活動をオンラインで行う必要が生じています。オンライン授業の実施には、学生と教員の ID 管理が不可欠となります。組織内での ID 管理の仕組みの重要性が改めて注目されました。また、クラウドサービスの活用が大規模に進みました。教育におけるクラウド利用と認証連携について、改めて考える必要が出てきました。今回のシンポジウムは、認証や認証連携が鍵となりますが、広く教育・研究のオンライン化に資するものとなると期待しています。“となっている。参加者については約 100 名ほどであった。

テーマとしては、“今後大学内での認証基盤をどのように作っていくか “、”NII の学認プロジェクトも新しい方向に向かおうとしている中で、ソフトウェアを中心とするシステム開発をうまくやるには、非サイロ化を目指さないといけないが、言うは易し行うは難し。“であり、本研究で取り組んでいる認証認可基盤全体のフ

レームワーク化、認証認可基盤の管理運用の簡略化について発表を行い議論を行い、本研究の認証認可基盤および管理運用統合プレーンの共有を行った。

これらを踏まえて本研究の属している分野のコミュニティに対しての貢献とする。

第 7 章
認証認可基盤の管理運用についての標準化に向けた提言

7.1. 標準化の概要

本研究の成果をもとに認証認可基盤の管理運用を行うための標準について提言をまとめる。また本提言については認証認可基盤だけではなく複数の要素およびソフトウェアからなるシステムに対しても有効な手段となり得る。

本標準で提供する機能の全体図を図 7.1 に示す。本標準は、管理運用統合プレーン、認証認可基盤における機能、認証認基盤から管理運用統合プレーンへの管理情報の送信機能、管理運用統合プレーンから認証認可基盤への運用の適用機能で構成されている。次節からそれぞれの機能の詳細とアーキテクチャ、データ構造、プロトコルについて示す。

7.2. 管理運用統合プレーン、制御プレーンの役割

管理運用統合プレーンの役割は、運用者に対する管理運用が必要となる管理情報の提供と認証認可基盤の各モジュールへの運用に必要となるアクションの実施である。そのため管理運用統合プレーンでは運用者が直感的に理解しやすい形式での情報の提供と直感的に実行可能なアクションの提供が行われるべきである。モジュール内の制御プレーンの役割はシステム、プログラムリーダブルな、管理情報の元となる各種データとアクションを構成しているアプリケーションリーダブルな一連の処理の実行である。そのためモジュールとソフトウェアで行われる

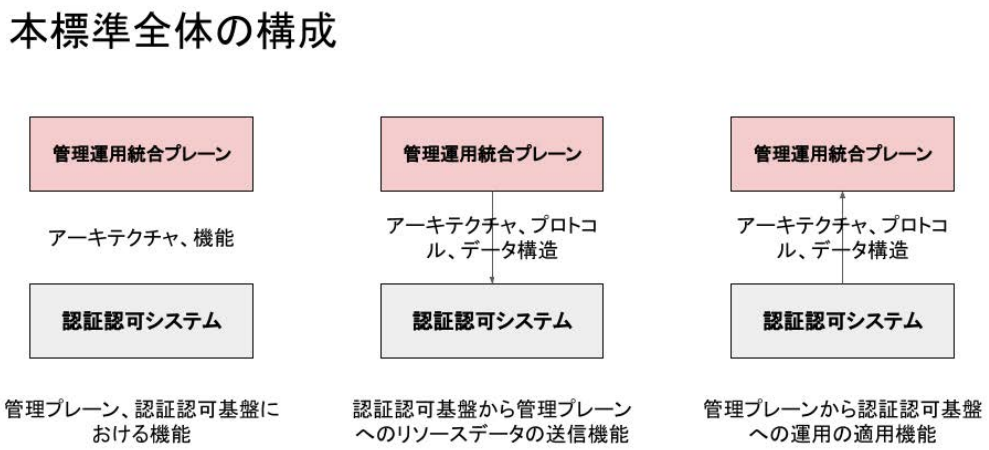

図 7.1 本標準全体の構成

逐次処理を正確にソフトウェアに対して再現可能な操作記法で記述するべきである。各プレーンの役割と処理を管理情報である CPU 使用率提示の例として図 7.2 に、IdP に対して新規ユーザーを追加するアクションの例として図 7.3 に示す。

7.3. 認証認可基盤モジュール

認証認可基盤で利用するモジュールについて定義する。モジュールは認証認可基盤を構成するための機能の最小単位であり、モジュールの機能に合わせて自由にソフトウェアをモジュール内に実装することが可能ある。モジュールにはソフトウェアに対して統合的に運用を実施する運用ファンクションとソフトウェアから管理情報を取得し、管理運用統合プレーンで一元管理するためのフォーマットに整形する管理情報の定義機能で構成されている。運用ファンクションでは付録 H に示すように、各モジュールで必要となる運用を定義することでソフトウェアの操作方法に左右されず管理運用統合プレーンから一元的な運用を可能とする。管理情報の定義機能では付録 G に示すように、管理運用統合プレーンで管理する管

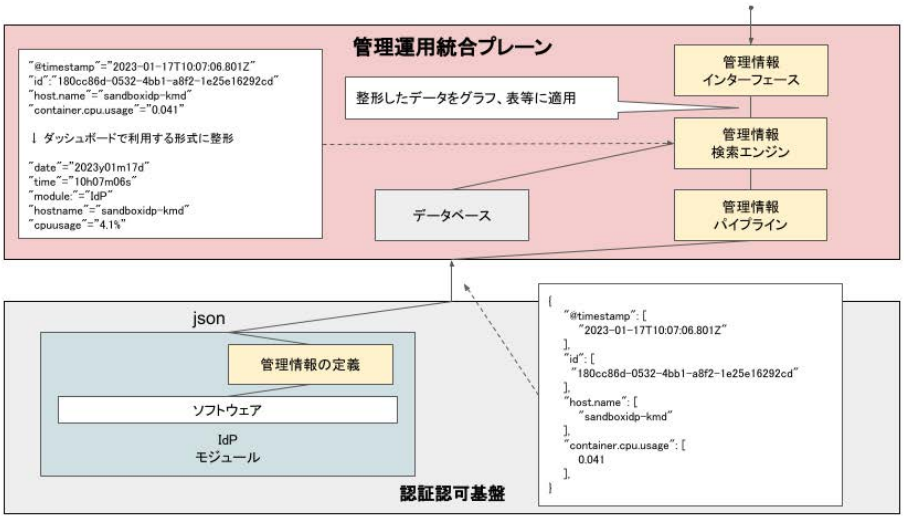

図 7.2　CPU 使用率提示の例

理情報を定義することソフトウェアの出力する管理情報フォーマットに依存することなく一元的な管理情報の把握を行うことを実現する。ただしこの際にソフトウェアには最低限出力する管理情報を定義し実装する必要がある。またこれらは付録 H、G に示した定義形式によって同様に各組織に合わせた設計が可能である。

7.4. 管理運用統合プレーン、認証認可基盤における機能

管理運用統合プレーン、認証認可基盤における機能を定義する。

7.4.1 アーキテクチャ

管理運用統合プレーン、認証認可基盤の全体のアーキテクチャを図 7.4 に示す。管理情報に関する要素を黄色、運用実行に関する要素を紫色で示している。

管理運用統合プレーンは、認証認可基盤の管理情報を取り扱う機能と認証認可基盤に運用を実行するための機能、また管理運用統合プレーンで発生するログ、

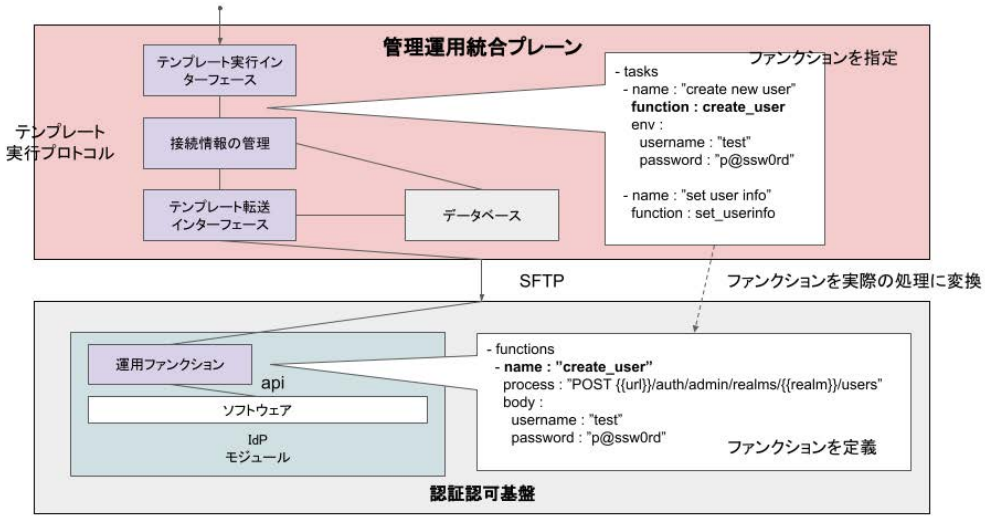

図 7.3　新規ユーザーを追加するアクションの例

認証情報、テンプレートを保存するためのデータベースで構成する。認証認可基盤は、TLS を実現する SSL Proxy モジュール、認証認可機能を実現する IdP モジュール、認証認可に必要な情報を保存する DB モジュールで構成されている。モジュールには、各機能を実現するためのソフトウェア、例えば、SSL Proxy モジュールでは、Nginx、Squid、IdP モジュールでは、Keycloak、OpenAM、DM モジュールでは、MySQL、PostgreSQL と、各ソフトウェアから出力される管理情報を定義し整形する機能、各ソフトウェアに対して実行する操作を定義しているファンクションから運用をを実行する運用ファンクションで構成する。ここでいうモジュールとはコンテナ技術を用いて分離すること、もしくはソフトウェアに運用ファンクションと管理情報の定義を実装することである。二つの場合は同意である。

7.4.2　機能

管理運用統合プレーンで必要となる機能は、以下の通りである。

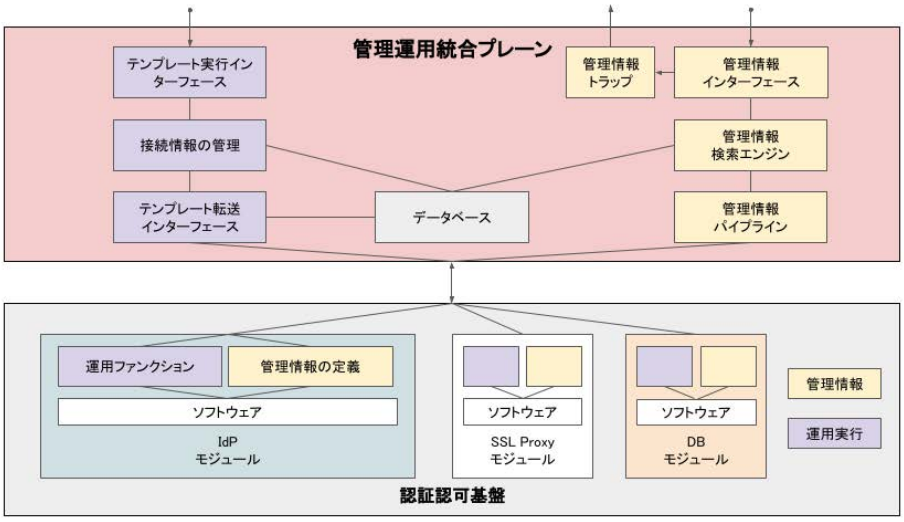

図 7.4　管理運用統合プレーン、認証認可基盤における機能と全体のアーキテクチャ

- 管理情報インターフェース
  - 管理情報の収集
  - 管理情報の可視化
  - 管理情報トラップ
- 運用実行インターフェース
  - 運用対象への接続情報管理
  - テンプレート管理
  - テンプレート実行

管理情報の収集

管理情報の収集に必要となる要素は、管理情報を認証認可基盤から受け付ける管理情報パイプライン、管理情報を保存するデータベースで構成されている。こ

れらの要素によって、認証認可基盤で生成され送信される管理情報をデータベースに蓄積する。

管理情報の可視化

管理情報の可視化に必要となる要素は、管理情報を検索する検索エンジン、管理情報を管理者に提供するインターフェースで構成されている。これらの要素によって、運用者にデータベースに蓄積された管理情報のダッシュボードでの可視化、管理情報の検索をする。

管理情報トラップ

管理情報トラップは管理情報を管理者に提供するインターフェースに構成されている。これにより、運用者に異常検知とサードパーティコミュニケーションツールに通知をする。

運用対象への接続情報管理

運用対象への接続情報管理に必要となる要素は、運用対象の接続情報の管理、接続に必要な機密情報の管理で構成されている。これらの要素によって、管理対象となるモジュールにアクセスする。

テンプレート管理

テンプレート管理に必要となる要素は、テンプレートを転送するインターフェース、テンプレートの保存を行うデータベースで構成されている。これらの要素によって、実行するテンプレートの対象モジュールもしくはデータベースへの転送とデータベースへの保存をする。

テンプレート実行

テンプレート実行に必要となる要素は、保存されているテンプレートの呼び出し、実行を行うインターフェースで構成されている。これらの要素によって、運用者から実行するテンプレートの転送もしくはデータベースからテンプレートを呼び出し運用を開始する。

認証認可基盤で必要となる機能は、以下の通りである。

- 管理情報インターフェース
  - 管理情報の生成
  - 管理情報の送信
- 運用実行インターフェース
  - テンプレート運用ファンクション
  - 拡張運用ファンクション

管理情報の生成

管理情報の生成に必要となる要素は、構成要素のモジュール、管理情報の定義で構成されている。これらの要素によって、ソフトウェアから出力されるログをモジュール内で定義されている管理情報のデータ構造に整形し管理情報の生成する。

管理情報の送信

管理情報の送信は管理情報の送信先を指定する設定と送信プロトコルで構成されている。これらの要素によって、モジュール内で生成された管理情報を送信先の設定をもとに管理情報パイプラインに対して送信する。

テンプレート運用ファンクション

テンプレート運用ファンクションは構成要素のモジュール毎の運用ファンクションの定義で構成されている。この要素によって、テンプレートで指定されているファンクションをソフトウェアの操作に置き換え、ソフトウェアに対して実行する。

拡張運用ファンクション

拡張運用ファンクションは構成要素のモジュールに対するテンプレートの編集機能で構成されている。この要素によって、テンプレートで実行可能となる新たなファンクションを定義する。

7.5. 認証認可基盤から管理運用統合プレーンへの管理情報の送信機能

認証認可基盤から管理運用統合プレーンへの管理情報の送信機能を定義する。イメージを図 7.5 に示す。

7.5.1 アーキテクチャ

認証認可基盤から管理運用統合プレーンへの管理情報の送信機能のアーキテクチャとしては、管理運用統合プレーンに管理情報インターフェース、管理情報トラップ、管理情報検索エンジン、管理情報パイプラインで構成される。認証認可基盤にはソフトウェアから出力される管理情報の定義と整形機能で構成される。

7.5.2 データ構造

認証認可基盤の管理情報のデータ構造を図 7.6 に示す。また実際のデータ構造および管理情報を付録 G に提示する。管理情報のデータ構造は、管理運用統合プレーン内での操作を高速に行うために json 形式とする。データ構造を実現するた

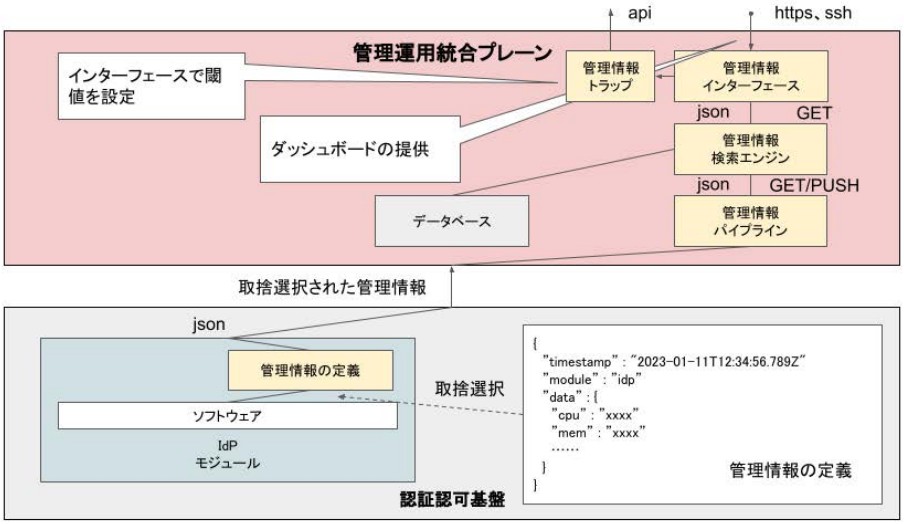

図 7.5　認証認可基盤から管理運用統合プレーンへの管理情報の送信機能

めに、あらかじめソフトウェアおよび管理情報の定義で用いる key の名前を定義しそれぞれ同じ名前を用いて管理情報を出力するように実装しなければならない。管理情報の定義では管理運用統合プレーンで必要になるデータを取捨選択し整形することができる。またこれらの管理情報の生成、出力頻度を設定できるべきである。

7.5.3　プロトコル

管理情報パイプラインは管理情報を受け取ると管理情報検索エンジンに対して http PUSH で送信する。管理情報検索エンジンは管理情報を受け取るとデータベースに json 形式で保存する。運用者に対して管理情報インターフェースでは、http でダッシュボードを提供する。ダッシュボードで管理情報が検索されると管理情報インターフェースから管理情報エンジンに http GET で管理情報を提供する。

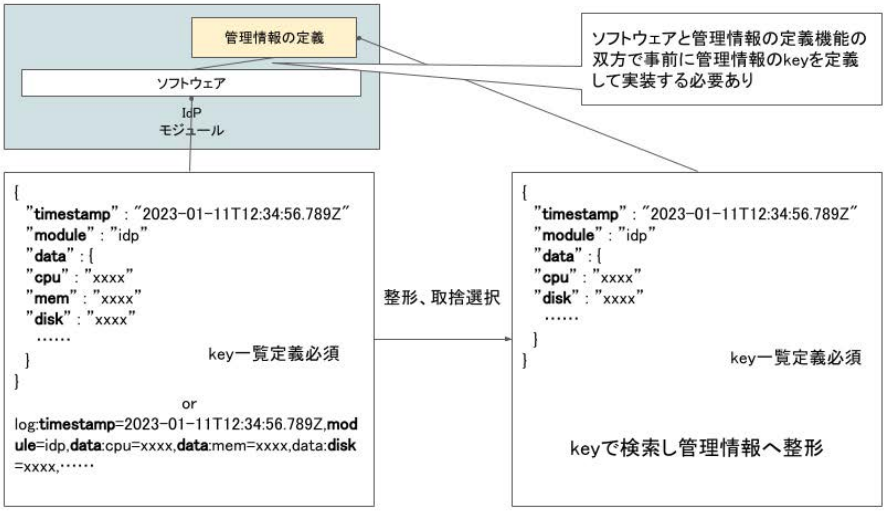

図 7.6　認証認可基盤の管理情報のデータ構造

7.6. 管理運用統合プレーンから認証認可基盤への運用の適用機能

管理運用統合プレーンから認証認可基盤への運用の適用機能を定義する。イメージを図 7.7 に示す。

7.6.1 アーキテクチャ

管理運用統合プレーンから認証認可基盤への運用の適用機能のアーキテクチャとしては、管理運用統合プレーンにテンプレート実行インターフェース、接続情報の管理、テンプレート転送インターフェースで構成される。また接続情報とテンプレートを管理するためのデータベースで構成される。認証認可基盤ではテンプレートの内容からソフトウェア操作のファンクションを実行する運用ファンクションで構成される。

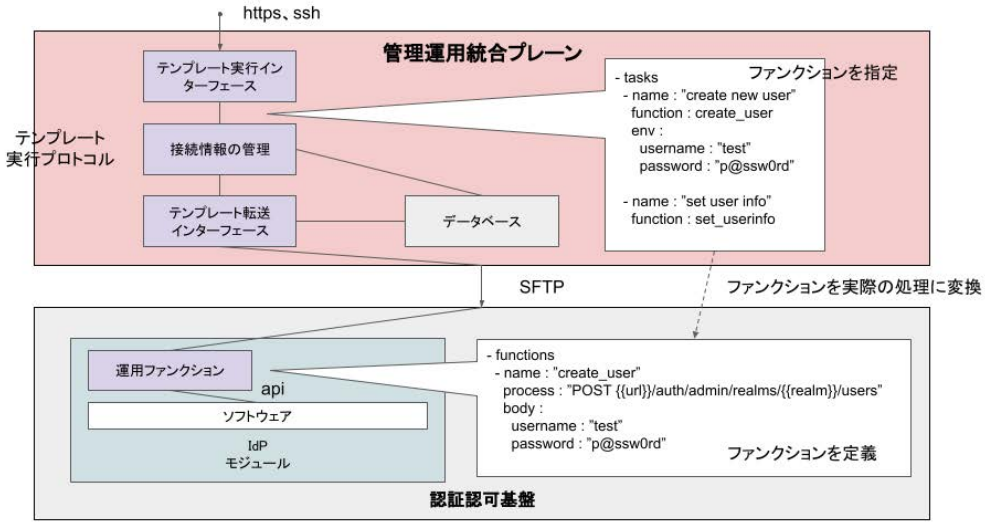

図 7.7　管理運用統合プレーンから認証認可基盤への運用の適用機能

7.6.2　データ構造

テンプレートのデータ構造とファンクションのデータ構造を図 7.6 に示す。また実際のデータ構造を付録 H に提示する。テンプレートのデータ構造は、yaml 形式、json 形式のような key-value の形式でかつ API で利用しやすい形式を取る必要がある。ここでは yaml 形式を用いて例を示す。

7.6.3　プロトコル

管理運用統合プレーンから認証認可基盤に対しては SFTP を利用してテンプレートを送信する。テンプレート実行インターフェースは運用者に http でダッシュボードを提供する。接続情報の管理機能はダッシュボードから事前に設定されるモジュールへの接続情報をデータベースに保存する。運用者がテンプレートをダッシュボードから実行する際、テンプレートに記載された対象となるモジュールの情報から接続情報の管理機能で対象となる接続情報を呼び出し、テンプレート転送インターフェースにテンプレートと認証情報を送信する。テンプレート転

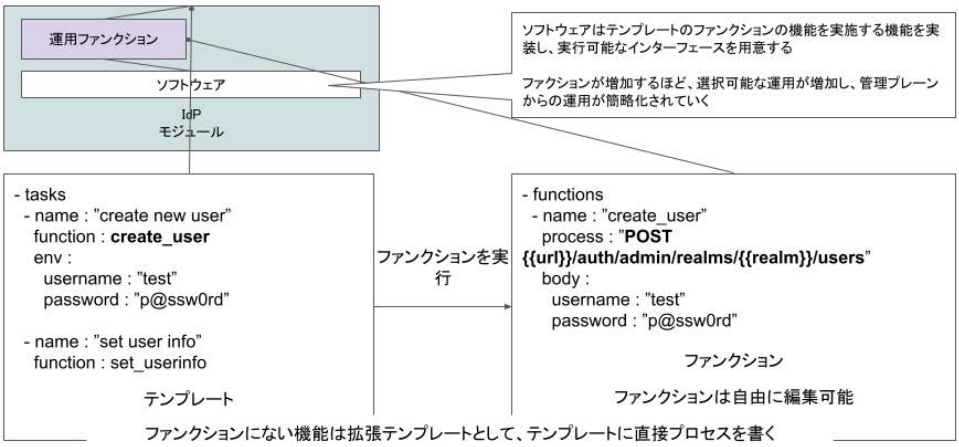

図 7.8 テンプレート、ファンクションのデータ構造

送インターフェースは受け取ったテンプレートを認証情報をもとに SFTP で対象となるモジュールに送信する。モジュールにテンプレートが送信されたことを検出すると、テンプレート転送インターフェースから SFTP を用いてテンプレートを実行するコマンドを送信する。

7.7. 本章のまとめ

以上により、認証認可システムの管理運用についての標準化に向けた提言とする。また本研究により設計された標準で用いる設定ファイルやプロトコルで利用するデータの要素、構成、ファイル形式、コード形態、データ量などの実運用環境の要件については、IOT コミュニティやシステム運用者コミュニティで議論するべきである。

第 8 章
結　　論

本論文では、実践的な認証認可基盤の管理運用を実現を目的とし、認証認可基盤および管理運用統合プレーンの設計、実装を行った。

2 章でも述べた通り、認証認可の役割は、何かしらのサービスを利用する際に、利用者がサービスに対して本人であることを証明し、サービスが本人確認をしたのちに利用者に対して適切なリソースを提供することであり、認証認可の仕組みがなければ今日のインターネットにおけるサービスを利用することはできない。しかし、認証認可を起因とする個人情報の漏洩などのインシデントは後を絶っていないのが現状である。この問題を解決するために、NIST を中心とする国際的標準化団体が Digital Identity に関するガイドラインを発行しているものの、認証認可に関する十分な人材と管理運用のノウハウがある組織でしかこれに適応できていないのが実情である。そこで本研究ではターゲットとして、自組織で認証認可基盤を管理運用する必要があるが、十分な人材も管理運用のノウハウがないもしくはすでに独自の認証認可基盤を構築している組織 (3.1 節の 3)、認証認可を行う必要があるが古い環境のまま行っており、これから認証認可基盤を導入しようとしている組織 (3.1 節の 4) を対象とし、取り扱う課題については、これらのターゲットが利用可能な自組織内における認証認可基盤の構築、管理運用の体系的なベストプラクティスが少ないということに注視し、認証認可基盤全体のフレームワーク化、認証認可基盤の管理運用の簡略化について取り組むこととした。

解決方法としては、3 章で述べた通り、課題を達成するための目標を実践的な認証認可基盤を実現として、そのために、認証認可基盤全体のフレームワーク化、認証認可基盤の管理運用の簡略化を行うこととした。まず認証認可基盤全体のフレームワーク化については、“環境への依存が少ないシステム構成 “、“再現性の

高い認証認可基盤のワンストップな構築 “、“継続的な管理運用が容易なシステムアーキテクチャ“の 3 つを解決するべき課題と考察し、これを解決することでターゲットとしている管理運用能力の乏しい組織環境に対して、実践的な認証認可基盤を提供し、尚且つ自身で容易に再現性をもって構築を行うことが可能となると考えた。次に認証認可基盤の管理運用の簡略化については、“認証認可基盤の状況把握 “、“認証認可基盤の管理運用の簡略化 “、“ユーザーアクセスとアカウントの状況把握 “、“アカウントマネジメントの簡略化 “の 4 つを解決するべき課題と考察し、認証認可基盤を構築しただけではなく、ノウハウや人材が十分ではないターゲットにおいても実際に管理運用を行うためのベストプラクティスとなることを目指した。

それぞれの設計および実装について述べる。環境への依存が少ないシステム構成については、4.7 節で示したアーキテクチャおよび 5.2 節のモジュール化の通り、それぞれの構成要素ごとに Docker によりコンテナ化することによって実現し、また同様に再現性の高い認証認可基盤のワンストップな構築についても、構築手順をコードで表現し実行することで同様な環境を構築可能とした。継続的な管理運用が容易なシステムアーキテクチャについては、図 4.10 に示すように管理運用統合プレーンをそれぞれのシステムに対して 5.3.2 項で示した IaC を用いた管理運用テンプレートを反映可能とすることで実現した。

認証認可基盤の状況把握については、5.2.2 項に示した各構成要素からのデータの取得、5.3.1 項のログ基盤による収集と可視化を行うことで実現している。認証認可基盤の管理運用の簡略化については、5.2 節による管理しやすい認証認可基盤の構成、5.3 節による管理運用統合プレーンによる管理の簡略化と運用のテンプレート化によって実現を図った。ユーザーアクセスとアカウントの状況把握については、4.8 節で設計したデータを 5.2.2 項の要件に合わせて認証認可基盤からアクセスログを取得し 5.3.1 項で構築するログダッシュボードを活用することによって実現した。アカウントマネジメントの簡略化については、ユーザーアクセスとアカウントの状況把握を行った後に 5.3.2 項、5.3.3 項で設計した運用テンプレートを用いることで実現を図った。

これらの設計および実装により本研究のターゲットに対して、認証認可基盤全

体のフレームワーク化、認証認可基盤の管理運用の簡略化を実現した。

ここまでの設計と実装を行った認証認可基盤および管理運用統合プレーンについて評価を行った。認証認可基盤全体のフレームワーク化の評価については、6.2.5 項、6.2.5 項で示した通り、用意した環境に対して 6.2.2 節で示した被験者に実際に構築してもらうことで評価とした。結果としては設計通りに認証認可基盤および管理運用統合プレーンをワンストップで行うことが可能であったが、構築者が構築時にすべてのソースコードを確認することは困難である問題が浮き彫りになり、結果として構築手順が増えてしまう結果となったが事前に環境変数を指定するファイルを用意したのちに構築を行う形式をとった。これにより継続的な管理運用が可能となる認証認可基盤および管理運用統合プレーンをターゲットが再現性をもって構築可能となるフレームワークを実現したと評価した。

本研究の成果から、管理運用統合プレーンを導入した認証認可基盤とその運用方法を標準化するための提言としてまとめた。これらの提言は今後本格的にシステムへの導入に向けて、IOT コミュニティやシステム運用者コミュニティで議論するべきであることを示した。

最後に本研究の貢献としては、学術分野としては IOT 分野への論文投稿と研究発表を行うことで認証認可基盤および管理運用統合プレーンのあり方について共有を行い、ヤングリサーチャ賞を受賞するに至った。また社会貢献としては、本研究の共同研究での成果および第 14 回統合認証シンポジウムでの本研究についての講演および議論を 260 以上の教育組織が参加する学認コミュニティに対して実施した。また本研究における認証認可基盤および管理運用統合プレーンのソースコードは Github にて公開しており、ライセンスの範囲内において自由に活用することが可能となっおり、さまざまな組織に対して、実践的な認証認可基盤を提供することで当該分野への貢献をすることができるであろう。

今後の展望としては、ますますインターネットにおいて認証認可は必要な要素となると考え、継続的な運用を前提とした認証認可基盤の設計と標準化によって、様々な組織において自由な認証認可基盤を構成するだけではなく、認証認可基盤を構成するソフトウェアが本標準に則って実装されることによる、継続的な認証認可基盤の管理運用が行える世界となるべきである。また運用の自動化や効率化

によって自組織内での運用のノウハウではなく、全運用者のノウハウとなる世界を目指し本研究が貢献されることを期待する。

業 績 リ ス ト

学会発表

- 加藤大弥, 藤尾正和, 高橋健太, 林達也, 砂原秀樹. 学内サービスにおける多要素認証の導入における検討. マルチメディア分散協調とモバイルシンポジウム 2018 論文集, 第 2018 巻, pp.1673–1679, 2018 年 6 月
- ヤングリサーチャ賞受賞, 加藤大弥, 藤尾正和, 林達也, 砂原秀樹. 学内サービスパスワードレス化の実現性の検討. マルチメディア分散協調とモバイルシンポジウム 2019 論文集, 第 2019 巻, pp.1784–1789, 2019 年 6 月
- 加藤大弥, 砂原秀樹. 研究科単位での学内サービスの id 統一化に向けた idp の導入および運用についての検討. インターネットと運用技術シンポジウム論文集, 第 2021 巻, pp.17–23, 2021 年 11 月

対外発表

- “研究科単位での学内サービスの ID 統一化に向けた IdP の導入および運用についての検討 “、第 14 回統合認証シンポジウム、2022 年 3 月

謝　　辞

博士学位論文を提出するにあたって、多くの方々のご指導とご助力をいただきました。

本研究の主指導教員であり、日々の研究へのご指導だけではなく公私共に暖かく見守ってくださった慶應義塾大学大学院メディアデザイン研究科の砂原秀樹教授に心から感謝致します。日々気にかけてくださる姿に常に励まされました。

副指導教員であり、主査を引き受けて頂きました慶應義塾大学大学院メディアデザイン研究科の加藤朗教授に心から感謝致します。技術的な面で常にリードし鋭いご指摘とアイディアを頂きました。

副指導教員であり、副査を引き受けて頂きました慶應義塾大学大学院メディアデザイン研究科の南澤孝太教授に心から感謝致します。指導教員として、また頼れる仲間としてKMDで一番長い期間ご指導を頂きました。

副査を引き受けて頂きました慶應義塾大学大学院メディアデザイン研究科の大川恵子教授に心から感謝致します。8年という長い学生生活を親身にサポートして頂きました。

副査を引き受けて頂きました株式会社日立製作所の鍛忠司博士に心から感謝致します。共同研究で右も左もわからない私を6年間正しい方向へと導いて頂きました。

学生生活、教員生活において多大なご協力を頂いた、慶應義塾大学大学院メディアデザイン研究科のNetwork Media Project、Embodied Media Project所属の教職員、研究者、学生、スタッフの方々に深く感謝致します。特に慶應義塾大学大学院メディアデザイン研究科後期博士課程に共に進学し苦楽を分かち合った同期の仲間達に心から感謝致します。みなさんの暖かい励ましが常に私に光をもたらしてくれました。

東京電機大学情報環境学部宮保研究室の皆さんに心から感謝致します。私の現在の進路への礎を築いたのは紛れもなく皆さんと寝食を忘れて過ごしたかけがえのない 2 年間のおかげです。

私を長く支えてくれたかけがえのない二人の友人に感謝致します。将来のことなど何も考えずに毎日遊び呆けていた高校生活から何も変わらずに現在まで笑って生活できたのはあなたのおかげです。本当にありがとう。

最後にこれまでの 30 年間暖かく支え、絶え間なく支援し、叱咤激励と愛を注いでくれた家族に心から感謝致します。

参 考 文 献

[1] Digital Agency. デジタル庁. https://www.digital.go.jp/. (2022 年 09 月 02 日閲覧).

[2] Digital Agency. e-gov 個人情報の保護に関する法律. https://elaws.e-gov.go.jp/document?lawid=415AC0000000057. (2022 年 09 月 02 日閲覧).

[3] Information technology Promotion Agency. 情報セキュリティ10 大脅威 2022. https://www.ipa.go.jp/security/vuln/10threats2022.html. (2022 年 09 月 02 日閲覧).

[4] 7pay. 7 pay サービス終了に伴うお詫びとお知らせ. https://www.7pay.co.jp/. お知らせ一覧 2019 年 (2019 年 11 月 12 日閲覧).

[5] Marriott International. Starwood guest reservation database security incident. https://info.starwoodhotels.com/. Starwood Guest Reservation Database Security Incident (2019 年 11 月 12 日閲覧).

[6] 7pay. 「7pay（セブンペイ）」 サービス廃止のお知らせとこれまでの経緯、今後の対応に関する説明について. https://www.7andi.com/library/dbps_data/_template_/_res/news/2019/20190801_01.pdf. お知らせ一覧 2019 年 (2020 年 2 月 6 日閲覧).

[7] ISO. Iso/iec 24760-1:2019 it security and privacy — a framework for identity management — part 1: Terminology and concepts. https://www.iso.org/obp/ui/#iso:std:iso-iec:24760:-1:ed-2:v1:en. (2022 年 09 月 02 日閲覧).

[8] J-LIS. 地方公共団体情報システム機構 マイナンバーカード 総合サイト. https://www.kojinbango-card.go.jp/card/. (2022 年 09 月 02 日閲覧).

[9] Information technology Promotion Agency. オンライン本人認証方式の実態調査 報告書. https://www.ipa.go.jp/files/000040778.pdf. (2022 年 09 月 02 日閲覧).

[10] National Institute of Standards and Technology. Nist special publication 800-63-3 digital identity guidelines. https://openid-foundation-japan.github.io/800-63-3-final/sp800-63-3.ja.html. NIST Special Publication 800-63-3 Digital Identity Guidelines(2019 年 11 月 16 日閲覧).

[11] Matt Weir, Sudhir Aggarwal, Michael Collins, and Henry Stern. Testing metrics for password creation policies by attacking large sets of revealed passwords. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, p. 162–175, New York, NY, USA, 2010. Association for Computing Machinery. URL: https://doi.org/10.1145/1866307.1866327, doi:10.1145/1866307.1866327.

[12] 総務省. 国民のための情報セキュリティサイト 安全なパスワード管理. https://www.soumu.go.jp/main_sosiki/cybersecurity/kokumin/business/business_staff_01.html. 国民のための情報セキュリティサイト (2022 年 09 月 02 日閲覧).

[13] GOOGLE CLOUD. Notifying administrators about unhashed password storage. https://cloud.google.com/blog/products/g-suite/notifying-administrators-about-unhashed-password-storage. (2022 年 09 月 02 日閲覧).

[14] 警察庁生活安全局情報技術犯罪対策課. 不正アクセス行為対策等の実態調査. https://www.npa.go.jp/cyber/research/r3/R3countermeasures.pdf. (2022 年 09 月 02 日閲覧).

[15] GOOGLE. Google アカウントを使用して他のアプリやサービスにログインする. https://support.google.com/accounts/answer/112802?hl=ja&co=GENIE.Platform. (2022 年 09 月 02 日閲覧).

[16] Twitter. 認証 twitter でログイン. https://developer.twitter.com/ja/docs/authentication/guides/log-in-with-twitter. (2022 年 09 月 02 日閲覧).

[17] 国立情報科学研究所. 学認 gakunin. https://www.gakunin.jp/. 学認 GakuNin(2019 年 11 月 23 日閲覧).

[18] Microsoft. Azure active directory. https://azure.microsoft.com/ja-jp/products/active-directory/. (2022 年 09 月 02 日閲覧).

[19] OASIS. Oasis saml wiki. https://wiki.oasis-open.org/security/FrontPage. OASIS SAML Wiki(2019 年 11 月 21 日閲覧).

[20] OpenID Foundation. Openid certification. https://openid.net/certification/. OpenID Certification(2019 年 12 月 21 日閲覧).

[21] OpenID Foundation. Openid connect core 1.0. https://openid.net/specs/openid-connect-core-1_0.html.

[22] OpenID Foundation. Openid connect discovery 1.0. https://openid.net/specs/openid-connect-discovery-1_0.html.

[23] OpenID Foundation. Openid connect dynamic client registration 1.0. https://openid.net/specs/openid-connect-registration-1_0.html.

[24] OpenID Foundation. Oauth 2.0 multiple response type encoding practices. https://openid.net/specs/oauth-v2-multiple-response-types-1_0.html.

[25] OpenID Foundation. Oauth 2.0 form post response mode. https://openid.net/specs/oauth-v2-form-post-response-mode-1_0.html.

[26] OpenID Foundation. Openid connect rp-initiated logout 1.0. https://openid.net/specs/openid-connect-rpinitiated-1_0.html.

[27] OpenID Foundation. Openid connect session management 1.0. https://openid.net/specs/openid-connect-session-1_0.html.

[28] OpenID Foundation. Openid connect front-channel logout 1.0. https://openid.net/specs/openid-connect-frontchannel-1_0.html.

[29] OpenID Foundation. Openid connect back-channel logout 1.0. https://openid.net/specs/openid-connect-backchannel-1_0.html.

[30] OpenID Foundation. Openid connect basic client implementer's guide 1.0. https://openid.net/specs/openid-connect-basic-1_0.html.

[31] OpenID Foundation. Openid connect implicit client implementer's guide 1.0. https://openid.net/specs/openid-connect-implicit-1_0.html.

[32] Dick Hardt. The OAuth 2.0 Authorization Framework. RFC 6749, October 2012. URL: https://www.rfc-editor.org/info/rfc6749, doi:10.17487/RFC6749.

[33] Michael Jones and Dick Hardt. The OAuth 2.0 Authorization Framework: Bearer Token Usage. RFC 6750, October 2012. URL: https://www.rfc-editor.org/info/rfc6750, doi:10.17487/RFC6750.

[34] Brian Campbell and Hannes Tschofenig. An IETF URN Sub-Namespace for OAuth. RFC 6755, October 2012. URL: https://www.rfc-editor.org/info/rfc6755, doi:10.17487/RFC6755.

[35] Torsten Lodderstedt, Mark McGloin, and Phil Hunt. OAuth 2.0 Threat Model and Security Considerations. RFC 6819, January 2013. URL: https://www.rfc-editor.org/info/rfc6819, doi:10.17487/RFC6819.

[36] Torsten Lodderstedt, Stefanie Dronia, and Marius Scurtescu. OAuth 2.0 Token Revocation. RFC 7009, August 2013. URL: https://www.rfc-editor.org/info/rfc7009, doi:10.17487/RFC7009.

[37] Michael Jones, John Bradley, and Nat Sakimura. JSON Web Token (JWT). RFC 7519, May 2015. URL: https://www.rfc-editor.org/info/rfc7519, doi:10.17487/RFC7519.

[38] Brian Campbell, Chuck Mortimore, Michael Jones, and Yaron Y. Goland. Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants. RFC 7521, May 2015. URL: https://www.rfc-editor.org/info/rfc7521, doi:10.17487/RFC7521.

[39] Brian Campbell, Chuck Mortimore, and Michael Jones. Security Assertion Markup Language (SAML) 2.0 Profile for OAuth 2.0 Client Authentication and Authorization Grants. RFC 7522, May 2015. URL: https://www.rfc-editor.org/info/rfc7522, doi:10.17487/RFC7522.

[40] Michael Jones, Brian Campbell, and Chuck Mortimore. JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants. RFC 7523, May 2015. URL: https://www.rfc-editor.org/info/rfc7523, doi:10.17487/RFC7523.

[41] Justin Richer, Michael Jones, John Bradley, Maciej Machulak, and Phil Hunt. OAuth 2.0 Dynamic Client Registration Protocol. RFC 7591, July 2015. URL: https://www.rfc-editor.org/info/rfc7591, doi:10.17487/RFC7591.

[42] Justin Richer, Michael Jones, John Bradley, and Maciej Machulak. OAuth 2.0 Dynamic Client Registration Management Protocol. RFC 7592, July 2015. URL: https://www.rfc-editor.org/info/rfc7592, doi:10.17487/RFC7592.

[43] Nat Sakimura, John Bradley, and Naveen Agarwal. Proof Key for Code Exchange by OAuth Public Clients. RFC 7636, September 2015. URL: https://www.rfc-editor.org/info/rfc7636, doi:10.17487/RFC7636.

[44] Justin Richer. OAuth 2.0 Token Introspection. RFC 7662, October 2015. URL: https://www.rfc-editor.org/info/rfc7662, doi:10.17487/RFC7662.

[45] Michael Jones, John Bradley, and Hannes Tschofenig. Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs). RFC 7800, April 2016. URL: https://www.rfc-editor.org/info/rfc7800, doi:10.17487/RFC7800.

[46] Michael Jones, Phil Hunt, and Anthony Nadalin. Authentication Method Reference Values. RFC 8176, June 2017. URL: https://www.rfc-editor.org/info/rfc8176, doi:10.17487/RFC8176.

[47] William Denniss and John Bradley. OAuth 2.0 for Native Apps. RFC 8252, October 2017. URL: https://www.rfc-editor.org/info/rfc8252, doi:10.17487/RFC8252.

[48] Michael Jones, Nat Sakimura, and John Bradley. OAuth 2.0 Authorization Server Metadata. RFC 8414, June 2018. URL: https://www.rfc-editor.org/info/rfc8414, doi:10.17487/RFC8414.

[49] William Denniss, John Bradley, Michael Jones, and Hannes Tschofenig. OAuth 2.0 Device Authorization Grant. RFC 8628, August 2019. URL: https://www.rfc-editor.org/info/rfc8628, doi:10.17487/RFC8628.

[50] Michael Jones, Anthony Nadalin, Brian Campbell, John Bradley, and Chuck Mortimore. OAuth 2.0 Token Exchange. RFC 8693, January 2020. URL: https://www.rfc-editor.org/info/rfc8693, doi:10.17487/RFC8693.

[51] Brian Campbell, John Bradley, Nat Sakimura, and Torsten Lodderstedt. OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access

Tokens. RFC 8705, February 2020. URL: https://www.rfc-editor.org/info/rfc8705, doi:10.17487/RFC8705.

[52] Brian Campbell, John Bradley, and Hannes Tschofenig. Resource Indicators for OAuth 2.0. RFC 8707, February 2020. URL: https://www.rfc-editor.org/info/rfc8707, doi:10.17487/RFC8707.

[53] Yaron Sheffer, Dick Hardt, and Michael Jones. JSON Web Token Best Current Practices. RFC 8725, February 2020. URL: https://www.rfc-editor.org/info/rfc8725, doi:10.17487/RFC8725.

[54] Vittorio Bertocci. JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens. RFC 9068, October 2021. URL: https://www.rfc-editor.org/info/rfc9068, doi:10.17487/RFC9068.

[55] Nat Sakimura, John Bradley, and Michael Jones. The OAuth 2.0 Authorization Framework: JWT-Secured Authorization Request (JAR). RFC 9101, August 2021. URL: https://www.rfc-editor.org/info/rfc9101, doi:10.17487/RFC9101.

[56] Torsten Lodderstedt, Brian Campbell, Nat Sakimura, Dave Tonge, and Filip Skokan. OAuth 2.0 Pushed Authorization Requests. RFC 9126, September 2021. URL: https://www.rfc-editor.org/info/rfc9126, doi:10.17487/RFC9126.

[57] Karsten Meyer zu Selhausen and Daniel Fett. OAuth 2.0 Authorization Server Issuer Identification. RFC 9207, March 2022. URL: https://www.rfc-editor.org/info/rfc9207, doi:10.17487/RFC9207.

[58] Michael Jones and Kristina Yasuda. JWK Thumbprint URI. RFC 9278, August 2022. URL: https://www.rfc-editor.org/info/rfc9278, doi:10.17487/RFC9278.

[59] Auth0. Token storage. https://auth0.com/docs/secure/security-guidance/data-security/token-storage.

[60] Auth0. Auth0. https://auth0.com/jp/. Auth0 Never Compromise on Identity (2019 年 11 月 15 日閲覧).

[61] Okta. The world ' s 1 identity platform okta. https://www.okta.com/.

[62] ITR CORPORATION. Itr market view：アイデンティティ・アクセス管理／個人認証型セキュリティ市場 2022. https://www.itr.co.jp/report/marketview/m22001000.html.

[63] 日本電子計算株式会社. 自治体専用 iaas システム「jip-base」の障害について. https://www.jip.co.jp/news/20191205/. ニュースリリース (2020 年 2 月 6 日閲覧).

[64] 学認. 学認アンケートを通して学ぶ正しい認証基盤構築ガイド. https://gakunin.jp/sites/default/files/2019-10/学認アンケートを通して学ぶ正しい認証基盤構築ガイド.pdf.

[65] 厚生労働省. 厚生労働省「シームレスな健康情報活用基盤実証事業」認証認可システムシステム仕様書. https://www.mhlw.go.jp/seisakunitsuite/bunya/kenkou_iryou/iryou/johoka/johokatsuyou/dl/tenpu03_03.pdf.

[66] NRI セキュアテクノロジーズ株式会社認. 認証認可の調査研究 最終報告書概要. https://www.mhlw.go.jp/content/12600000/000689498.pdf.

[67] NRI セキュアテクノロジーズ株式会社認. 認証認可の調査研究 最終報告書概要. https://www.mhlw.go.jp/content/12600000/000689498.pdf.

[68] The Apache Software Foundation. Apache http サーバ バージョン 2.4 認証、承認、アクセス制御. https://httpd.apache.org/docs/current/ja/howto/auth.html.

[69] Inc. VMware. Vmware esxi：専用のベアメタル ハイパーバイザー. https://www.vmware.com/jp/products/esxi-and-esx.html.

[70] Docker Inc. Docker: Empowering app development for developers. https://www.docker.com/. Docker (2020 年 2 月 7 日閲覧).

[71] the containers organization. podman. https://podman.io/.

[72] Zabbix. Zabbix オフィシャル日本語サイト. https://www.zabbix.com/jp. Zabbix オフィシャル日本語サイト (2020 年 2 月 6 日閲覧).

[73] Elasticsearch B.V. Metricbeat 軽量メトリックシッパー. https://www.elastic.co/jp/beats/metricbeat.

[74] Elasticsearch B.V. Elasticsearch. https://www.elastic.co/jp/elasticsearch/.

[75] Inc. Amazon Web Services. Amazon cloudwatch. https://aws.amazon.com/jp/cloudwatch/.

[76] Google. Cloud monitoring. https://cloud.google.com/monitoring?hl=ja.

[77] Keycloak. Keycloak open source identity and access management for modern applications and services. https://www.keycloak.org/. Open Source Identity and Access Management For Modern Applications and Services(2019 年 11 月 23 日閲覧).

[78] Inc. Red Hat. Product documentation for red hat jboss enterprise application platform 7.4. https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/7.3/html/configuration_guide/undertow-configure-server.

[79] 学認. ユーザアクセスのロギング. https://meatwiki.nii.ac.jp/confluence/pages/viewpage.action?pageId=12158429.

[80] Okta Developer. System log api. https://developer.okta.com/docs/reference/api/system-log/.

[81] Elasticsearch B.V. Kibana — データを探索、可視化、分析. https://www.elastic.co/jp/kibana. Kibana (2020 年 2 月 6 日閲覧).

[82] RedHat. Ansible is simple it automation. https://www.ansible.com/. Ansible is Simple IT Automation (2020 年 2 月 6 日閲覧).

[83] Kazuya Suzuki, Kentaro Sonoda, Nobuyuki Tomizawa, Yutaka Yakuwa, Terutaka Uchida, Yuta Higuchi, Toshio Tonouchi, and Hideyuki Shimonishi. A survey on openflow technologies. *IEICE Transactions on Communications*, Vol. 97, No. 2, pp. 375–386, 2014.

[84] B.K. Lee and L.K. John. Npbench: a benchmark suite for control plane and data plane applications for network processors. In *Proceedings 21st International Conference on Computer Design*, pp. 226–233, 2003. doi:10.1109/ICCD.2003.1240899.

[85] Zuhran Khan Khattak, Muhammad Awais, and Adnan Iqbal. Performance evaluation of opendaylight sdn controller. In *2014 20th IEEE international conference on parallel and distributed systems (ICPADS)*, pp. 671–676. IEEE, 2014.

[86] Liehuang Zhu, Md Monjurul Karim, Kashif Sharif, Fan Li, Xiaojiang Du, and Mohsen Guizani. Sdn controllers: Benchmarking & performance evaluation. *arXiv preprint arXiv:1902.04491*, 2019.

[87] Keith McCloghrie and Marshall Rose. Management information base for network management of tcp/ip-based internets: Mib-ii. Technical report, 1991.

[88] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Open-

flow: enabling innovation in campus networks. *ACM SIGCOMM computer communication review*, Vol. 38, No. 2, pp. 69–74, 2008.

[89] Adrian Lara, Anisha Kolasani, and Byrav Ramamurthy. Network innovation using openflow: A survey. *IEEE communications surveys & tutorials*, Vol. 16, No. 1, pp. 493–512, 2013.

[90] Prometheus Authors. Prometheus - monitoring system & time series database. `https://prometheus.io/`. Prometheus (2020 年 2 月 6 日閲覧).

[91] Istio Authors. Istio. `https://istio.io/`. Istio (2020 年 2 月 6 日閲覧).

[92] Prometheus Authors 2014-2020. Exporters and integrations. `https://prometheus.io/docs/instrumenting/exporters/`. Prometheus (2020 年 2 月 11 日閲覧).

[93] IETF. Automated network management - ietf. `https://ietf.org/topics/netmgmt/`. IETF (2020 年 2 月 6 日閲覧).

[94] Cloud Native Computing Foundation. Building sustainable ecosystems for cloud native software. `https://www.cncf.io/`. Cloud Native Computing Foundation(2019 年 11 月 25 日閲覧).

[95] Digital Agency. デジタル庁情報システム調達改革検討会（第 2 回）. `https://www.digital.go.jp/councils/b2cd2414-9e37-444b-93d7-1df08e3d1523/`.

[96] Ministry of Economy. オープンソースソフトウェアの利活用及びそのセキュリティ確保に向けた管理手法に関する事例集を取りまとめました. `https://www.meti.go.jp/press/2021/04/20210421001/20210421001.html`.

[97] Linux Kernel Organization. The linux kernel archives. `https://www.kernel.org/`.

[98] Linux Kernel Organization. The linux kernel archives. https://www.kernel.org/.

[99] Inc. Red Hat. Red hat enterprise linux. https://www.redhat.com/en/technologies/linux-platforms/enterprise-linux.

[100] Black Duck. Open hub. https://www.openhub.net/. Discover, Track and Compare Open Source(2019 年 12 月 21 日閲覧).

[101] Kangjie Lu. An open letter to the linux community. https://lore.kernel.org/lkml/CAK8KejpUVLxmqp026JY7x5GzHU2YJLPU8SzTZUNXU2OXC70ZQQ@mail.gmail.com/.

[102] ITmedia Inc. Oss「faker.js」と「colors.js」の開発者、自身でライブラリを意図的に改ざん 「ただ働きはもうしない」. https://togetter.com/li/1702680.

[103] keycloak/keycloak. Keycloak. https://github.com/keycloak/keycloak.

[104] The PostgreSQL Global Development Group. Postgresql downloads. https://www.postgresql.org/download/.

[105] SteveLTN/https-portal. Https-portal. https://github.com/SteveLTN/https-portal.

[106] Elasticsearch B.V. Kibana. https://www.elastic.co/jp/kibana/.

[107] Elasticsearch B.V. Metricbeat. https://www.elastic.co/jp/beats/metricbeat.

[108] fluent/fluentd. Fluentd: Open-source log collector. https://github.com/fluent/fluentd.

[109] radoondas/elasticbeat. metricbeat.template.json. https://github.com/radoondas/elasticbeat/blob/master/vendor/github.com/elastic/beats/metricbeat/metricbeat.template.json.

[110] Docker Inc. Configure logging drivers. https://docs.docker.com/config/containers/logging/configure/.

[111] Inc. Amazon Web Services. Analytics on aws. https://aws.amazon.com/big-data/datalakes-and-analytics/?nc1=h_ls.

[112] Inc. MaxMind. Geoip databases & services: Industry leading ip intelligence. maxmind.com/en/geoip2-services-and-databases.

[113] ansible/awx. Awx. https://github.com/ansible/awx.

[114] TOYOTA MOTOR CORPORATION. お客様のメールアドレス等の漏洩可能性に関するお詫びとお知らせについて. https://global.toyota/jp/newsroom/corporate/38095972.html.

[115] Arm Limited. arm プロセッサー. https://www.arm.com/ja/products/silicon-ip-cpu.

[116] Docker Inc. Install docker desktop on windows. https://docs.docker.com/desktop/install/windows-install/.

付　　録

本付録に含まれているコードは、Github(https://github.com/daiyak) において公開されている各種リポジトリに紐づいている。各種問い合わせについては同じく公開されている連絡先にすること。

A. 認証認可基盤の設計コード

Dockerfile (IdP の設計図)

```
FROM jboss/keycloak

# Overwrite welcome page
ADD themes/welcome /opt/jboss/keycloak/themes/keycloak/welcome
```

docker-compose.yml (認証認可基盤全体の設計図)

```
version: "3"
services:
  https-portal:
    image: steveltn/https-portal:1
    ports:
      - '80:80'
      - '443:443'
    links:
      - keycloak
    restart: always
    environment:
      DOMAINS: '${SSL_DOMAIN} -> http://keycloak:8080'
```

```
      STAGE: '${SSL_STAGING}'
      # FORCE_RENEW: 'true'
    volumes:
      - ../https-portal-data:/var/lib/https-portal
  keycloak:
    #image: jboss/keycloak
    build: keycloak
    # log の permition 問題のため root user で実行
    user: "0:0"
    restart: always
    ports:
      - "8080:8080"
    environment:
      - "KEYCLOAK_USER=${KEYCLOAK_USER:-admin}"
      - "KEYCLOAK_PASSWORD=${KEYCLOAK_PASSWORD:-komediadesign}"
      - "PROXY_ADDRESS_FORWARDING=true"
      # - "KEYCLOAK_LOGLEVEL=DEBUG"
      # - "ACL_IP=172.16.0.0/16"
      # - "SSL_DOMAIN=www.example.com"
      # - "SSL_STAGING=production"
      - "DB_VENDOR=${DB_VENDOR:-h2}"
      - "DB_ADDR=${DB_ADDR}"
      - "DB_DATABASE=${DB_DATABASE}"
      - "DB_USER=${DB_USER}"
      - "DB_PASSWORD=${DB_PASSWORD}"
```

B. コンテナ毎で取得可能な情報

- key : value
- cpu : CPU usage
- load : CPU load averages
- memory : Memory usage

- network : Network IO
- process : Per process metrics
- process_summary : Process summary
- uptime : System Uptime
- socket_summary : Socket summary
- core : Per CPU core usage
- diskio : Disk IO
- filesystem : File system usage for each mountpoint
- fsstat : File system summary metrics
- socket : Sockets and connection info

C. 定義したリソースデータのデータ構造の例

リソースデータの定義

```
{
    "metrics":{
        "container":"example_val",
        "cpu":"example_val",
        "diskio":"example_val",
        "healthcheck":"example_val",
        "info":"example_val",
        "memory":"example_val",
        "network":"example_val",
    },
    "logs":"syslog"
    "period":"10s"
}
```

出力されるリソースデータのデータ構造 - cpu 使用率の例


```
{
    "@timestamp": [
        "2023-01-17T10:07:06.801Z"
    ],
    "agent.ephemeral_id": [
        "8eef2809-21a4-4c91-940b-9e0b2630d2f0"
    ],
    "agent.hostname": [
        "sandboxidp-kmd"
    ],
    "agent.id": [
        "180cc86d-0532-4bb1-a8f2-1e25e16292cd"
    ],
    "agent.name": [
        "sandboxidp-kmd"
    ],
    "agent.type": [
        "metricbeat"
    ],
    "agent.version": [
        "8.3.3"
    ],
    "beats_state.state.host.name": [
        "sandboxidp-kmd"
    ],
    "beats_state.timestamp": [
        "2023-01-17T10:07:06.801Z"
    ],
    "container.cpu.usage": [
        0.041
    ],
    "container.id": [
        "7ce8c55efa2c33f6be754f65a4b9365d574cce30d89d2ebd23105a7dcca9a14a"
    ],
```

```
"container.image.name": [
    "docker.elastic.co/beats/metricbeat:8.3.3"
],
"container.name": [
    "idp_metricbeat_1"
],
"container.runtime": [
    "docker"
],
"docker.container.labels.com_docker_compose_config-hash": [
    "12c983f69bf31473d7e6ae5c37b1ab9fb6146df6422fa83204cba82c2fa38b3e"
],
"docker.container.labels.com_docker_compose_container-number": [
    "1"
],
"docker.container.labels.com_docker_compose_oneoff": [
    "False"
],
"docker.container.labels.com_docker_compose_project": [
    "idp"
],
"docker.container.labels.com_docker_compose_project_config_files": [
    "docker-compose.yml"
],
"docker.container.labels.com_docker_compose_project_environment_file": [
    "../idp_.env"
],
"docker.container.labels.com_docker_compose_project_working_dir": [
    "/home/daiya/idp"
],
"docker.container.labels.com_docker_compose_service": [
    "metricbeat"
],
"docker.container.labels.com_docker_compose_version": [
    "1.29.2"
],
```

```
"docker.container.labels.description": [
    "Metricbeat is a lightweight shipper for metrics."
],
"docker.container.labels.io_k8s_description": [
    "Metricbeat is a lightweight shipper for metrics."
],
"docker.container.labels.io_k8s_display-name": [
    "Metricbeat image"
],
"docker.container.labels.license": [
    "Elastic License"
],
"docker.container.labels.maintainer": [
    "infra@elastic.co"
],
"docker.container.labels.name": [
    "metricbeat"
],
"docker.container.labels.org_label-schema_build-date": [
    "2022-07-23T00:38:04Z"
],
"docker.container.labels.org_label-schema_license": [
    "Elastic License"
],
"docker.container.labels.org_label-schema_name": [
    "metricbeat"
],
"docker.container.labels.org_label-schema_schema-version": [
    "1.0"
],
"docker.container.labels.org_label-schema_url": [
    "https://www.elastic.co/beats/metricbeat"
],
"docker.container.labels.org_label-schema_vcs-ref": [
    "1755b5dd3127bf755ee39deb25a802438bdac620"
],
```

```
"docker.container.labels.org_label-schema_vcs-url": [
    "github.com/elastic/beats/v7"
],
"docker.container.labels.org_label-schema_vendor": [
    "Elastic"
],
"docker.container.labels.org_label-schema_version": [
    "8.3.3"
],
"docker.container.labels.org_opencontainers_image_created": [
    "2022-07-23T00:38:04Z"
],
"docker.container.labels.org_opencontainers_image_licenses": [
    "Elastic License"
],
"docker.container.labels.org_opencontainers_image_title": [
    "Metricbeat"
],
"docker.container.labels.org_opencontainers_image_vendor": [
    "Elastic"
],
"docker.container.labels.release": [
    "1"
],
"docker.container.labels.summary": [
    "metricbeat"
],
"docker.container.labels.url": [
    "https://www.elastic.co/beats/metricbeat"
],
"docker.container.labels.vendor": [
    "Elastic"
],
"docker.container.labels.version": [
    "8.3.3"
],
```

```
"docker.cpu.kernel.norm.pct": [
    0.022
],
"docker.cpu.kernel.pct": [
    0.045
],
"docker.cpu.kernel.ticks": [
    32379137648000
],
"docker.cpu.system.norm.pct": [
    1
],
"docker.cpu.system.pct": [
    2
],
"docker.cpu.system.ticks": [
    5161245570000000
],
"docker.cpu.total.norm.pct": [
    0.041
],
"docker.cpu.total.pct": [
    0.081
],
"docker.cpu.user.norm.pct": [
    0.018
],
"docker.cpu.user.pct": [
    0.037
],
"docker.cpu.user.ticks": [
    24624187822000
],
"ecs.version": [
    "8.0.0"
],
```

```
"event.dataset": [
    "docker.cpu"
],
"event.duration": [
    2049919440
],
"event.module": [
    "docker"
],
"host.name": [
    "sandboxidp-kmd"
],
"kibana_stats.timestamp": [
    "2023-01-17T10:07:06.801Z"
],
"logstash_stats.timestamp": [
    "2023-01-17T10:07:06.801Z"
],
"metricset.name": [
    "cpu"
],
"metricset.period": [
    10000
],
"service.address": [
    "unix:///var/run/docker.sock"
],
"service.type": [
    "docker"
],
"timestamp": [
    "2023-01-17T10:07:06.801Z"
],
"_id": "28kyv4UBM4JmurvuZS0r",
"_index": ".ds-metricbeat-8.3.3-2023.01.06-000005",
"_score": null
```

```
}
```

D. 定義したアクセスログのデータ構造の例

アクセスログフォーマットの例

```
{
    type=LOGIN,
    realmId=keio,
    clientId=idptest,
    userId=4a70fa0e-ccbb-4705-b583-e8b2210a1544,
    ipAddress=131.113.199.55,
    auth_method=openid-connect,
    auth_type=code,
    response_type=code,
    redirect_uri=http://canopus.kmd.keio.ac.jp/apps/
    sociallogin/custom_oidc/keio,
    consent=no_consent_required,
    code_id=8f27ba53-e474-4380-8e9c-d0dfc3abf30b,
    username=latte,
    response_mode=query,
    authSessionParentId=8f27ba53-e474-4380-8e9c-d0dfc3abf30b,
    authSessionTabId=iBqUV68hl8Y"
}
```

ログ生成の設定ファイル

```
embed-server --server-config=standalone-ha.xml --std-out=echo
# access log enabling
/subsystem=undertow/server=default-server/host=default-host/
setting=access-log:add

# access log size rotating
/subsystem=logging/size-rotating-file-handler=ACCESSLOG:
add(file={path=access_log.log,relative-to=jboss.server.log.dir})
/subsystem=logging/size-rotating-file-handler=ACCESSLOG:
```

```
write-attribute(name=level,value=INFO)
/subsystem=logging/size-rotating-file-handler=ACCESSLOG:
write-attribute(name=rotate-size, value=10M)
/subsystem=logging/size-rotating-file-handler=ACCESSLOG:
write-attribute(name=max-backup-index, value=5)
/subsystem=logging/size-rotating-file-handler=ACCESSLOG:
write-attribute(name=append,value=true)
/subsystem=logging/size-rotating-file-handler=ACCESSLOG:
write-attribute(name=formatter,value=
"%d{yyyy-MM-dd HH\:mm\:ss,SSS} %-5p [%c] (%t) %s%e%n")
/subsystem=logging/size-rotating-file-handler=ACCESSLOG:
write-attribute(name=autoflush,value=true)
/subsystem=logging/root-logger=ROOT:add-handler(name=ACCESSLOG)

# event log enabling
/core-service=management/access=audit/
logger=audit-log:write-attribute(name=enabled,value=true)

# event log setting
/subsystem=logging/logger=org.keycloak.events:add
/subsystem=logging/logger=org.keycloak.events:
write-attribute(name=level,value=DEBUG)

# event log size rotating
/subsystem=logging/size-rotating-file-handler=EVENTLOG:
add(file={path=events.log,relative-to=jboss.server.log.dir})
/subsystem=logging/size-rotating-file-handler=EVENTLOG:
write-attribute(name=level,value=DEBUG)
/subsystem=logging/size-rotating-file-handler=EVENTLOG:
write-attribute(name=rotate-size, value=10M)
/subsystem=logging/size-rotating-file-handler=EVENTLOG:
write-attribute(name=max-backup-index, value=5)
/subsystem=logging/size-rotating-file-handler=EVENTLOG:
write-attribute(name=append,value=true)
/subsystem=logging/size-rotating-file-handler=EVENTLOG:
write-attribute(name=formatter,value=
```

```
"%d{yyyy-MM-dd HH\:mm\:ss,SSS} %-5p [%c] (%t) %s%e%n")
/subsystem=logging/size-rotating-file-handler=EVENTLOG:
write-attribute(name=autoflush,value=true)
/subsystem=logging/root-logger=ROOT:add-handler(name=EVENTLOG)
```

実際に転送されるログの例


```
{
  "@log_name": [
    "docker.keycloak"
  ],
  "@log_name.keyword": [
    "docker.keycloak"
  ],
  "@timestamp": [
    "2022-09-14T07:08:57.700Z"
  ],
  "auth_method": [
    "openid-connect"
  ],
  "auth_method.keyword": [
    "openid-connect"
  ],
  "auth_type": [
    "code"
  ],
  "auth_type.keyword": [
    "code"
  ],
  "authSessionParentId": [
    "8f27ba53-e474-4380-8e9c-d0dfc3abf30b"
  ],
  "authSessionParentId.keyword": [
    "8f27ba53-e474-4380-8e9c-d0dfc3abf30b"
  ],
  "authSessionTabId": [
    "iBqUV68hl8Y"
```

```
  ],
  "authSessionTabId.keyword": [
    "iBqUV68hl8Y"
  ],
  "clientId": [
    "nextcloud-hitachi"
  ],
  "clientId.keyword": [
    "nextcloud-hitachi"
  ],
  "code_id": [
    "8f27ba53-e474-4380-8e9c-d0dfc3abf30b"
  ],
  "code_id.keyword": [
    "8f27ba53-e474-4380-8e9c-d0dfc3abf30b"
  ],
  "consent": [
    "no_consent_required"
  ],
  "consent.keyword": [
    "no_consent_required"
  ],
  "hostName": [
    "bf9dd6a94dd8"
  ],
  "hostName.keyword": [
    "bf9dd6a94dd8"
  ],
  "ipAddress": [
    "131.113.199.55"
  ],
  "ipAddress.keyword": [
    "131.113.199.55"
  ],
  "level": [
    "DEBUG"
```

```
  ],
  "level.keyword": [
    "DEBUG"
  ],
  "loggerClassName": [
    "org.jboss.logging.Logger"
  ],
  "loggerClassName.keyword": [
    "org.jboss.logging.Logger"
  ],
  "loggerName": [
    "org.keycloak.events"
  ],
  "loggerName.keyword": [
    "org.keycloak.events"
  ],
  "message": [
    "type=LOGIN, realmId=keio, clientId=idptest,
    userId=4a70fa0e-ccbb-4705-b583-e8b2210a1544,
    ipAddress=131.113.199.55,
    auth_method=openid-connect,
    auth_type=code, response_type=code,
    redirect_uri=http://canopus.kmd.keio.ac.jp/
    apps/sociallogin/custom_oidc/keio,
    consent=no_consent_required,
    code_id=8f27ba53-e474-4380-8e9c-d0dfc3abf30b,
    username=latte, response_mode=query,
    authSessionParentId=8f27ba53-e474-4380-8e9c-d0dfc3abf30b,
    authSessionTabId=iBqUV68hl8Y"
  ],
  "_id": "QmrUOoMB342R0oGeWd_s",
  "_index": "fluentd-20220914",
  "_score": null
}
```

E. ログ基盤での各種データの整形

アクセスログの整形

```
{
    type=LOGIN,
    realmId=keio,
    clientId=idptest,
    userId=4a70fa0e-ccbb-4705-b583-e8b2210a1544,
    ipAddress=131.113.199.55,
    auth_method=openid-connect,
    auth_type=code,
    response_type=code,
    redirect_uri=http://canopus.kmd.keio.ac.jp/apps/sociallogin/
                 custom_oidc/keio,
    consent=no_consent_required,
    code_id=8f27ba53-e474-4380-8e9c-d0dfc3abf30b,
    username=latte,
    response_mode=query,
    authSessionParentId=8f27ba53-e474-4380-8e9c-d0dfc3abf30b,
    authSessionTabId=iBqUV68hl8Y"
}
```

アクセスログの取得に伴う整形と index 化の config ファイル

```
<source>
    @type forward
    port 24224
    bind 0.0.0.0
</source>

<filter *.**>
    @type parser
    key_name log
    reserve_data false
    <parse>
        @type json
    </parse>
```

```
</filter>

<match *.**>
    @type copy
    <store>
        @type elasticsearch
        host elasticsearch
        port 9200
        logstash_format true
        logstash_prefix fluentd
        logstash_dateformat %Y%m%d
        include_tag_key true
        type_name access_log
        tag_key @log_name
        flush_interval 1s
        utc_index false
        enable_ilm true
    </store>

    <store>
        @type stdout
    </store>
</match>
```

アクセスログの定義

```
{
  "properties": {
    "@log_name": {
      "type": "text",
      "fields": {
        "keyword": {
          "type": "keyword",
          "ignore_above": 256
        }
      }
    },
```

```
"@timestamp": {
  "type": "date"
},
"hostName": {
  "type": "text",
  "fields": {
    "keyword": {
      "type": "keyword",
      "ignore_above": 256
    }
  }
},
"level": {
  "type": "text",
  "fields": {
    "keyword": {
      "type": "keyword",
      "ignore_above": 256
    }
  }
},
"loggerClassName": {
  "type": "text",
  "fields": {
    "keyword": {
      "type": "keyword",
      "ignore_above": 256
    }
  }
},
"loggerName": {
  "type": "text",
  "fields": {
    "keyword": {
      "type": "keyword",
      "ignore_above": 256
```

```
      }
    }
  },
  "mdc": {
    "type": "object"
  },
  "message": {
    "type": "text",
    "fields": {
      "keyword": {
        "type": "keyword",
        "ignore_above": 512
      }
    }
  },
  "ndc": {
    "type": "text",
    "fields": {
      "keyword": {
        "type": "keyword",
        "ignore_above": 256
      }
    }
  },
  "processId": {
    "type": "long"
  },
  "processName": {
    "type": "text",
    "fields": {
      "keyword": {
        "type": "keyword",
        "ignore_above": 256
      }
    }
  },
```

```
    "sequence": {
      "type": "long"
    },
    "threadId": {
      "type": "long"
    },
    "threadName": {
      "type": "text",
      "fields": {
        "keyword": {
          "type": "keyword",
          "ignore_above": 256
        }
      }
    },
    "timestamp": {
      "type": "date"
    }
  }
}
```

message 内のアクセスログの整形

```
[
  {
    "kv": {
      "field": "message",
      "field_split": ", ",
      "value_split": "="
    }
  },
  {
    "geoip": {
      "field": "ipAddress"
    }
  }
]
```

F. 管理運用統合プレーンで実際にコード化した作業

IdP におけるユーザー追加作業の例

ディレクトリ構造

```
- inventory
  - development.ini #実行環境の指定ファイル (開発環境用)
  - production.ini #実行環境の指定ファイル (本番環境用)
- roles
  - create_user
    - tasks
      - main.yml #実行する作業の定義ファイル
    - vars
      - main.yml #作業を実行するための環境変数の定義ファイル
- keycloak_create_user.yml #テンプレートファイル
```

```
development.ini
[localhost]
127.0.0.1
```

```
production.ini
[localhost]
127.0.0.1
```

```
tasks/main.yml
- name: "Create Token for service Keycloak"
  uri:
    url: "{{ keycloak_base_url }}/auth/realms/master/protocol/
        openid-connect/token"
    method: POST
    body_format: form-urlencoded
    body:
      username: "{{ keycloak_admin_user }}"
      password: "{{ keycloak_admin_pass }}"
      grant_type: "password"
      client_id: "admin-cli"
  register: keycloak_token
```

```
- name: "Debug var keycloak_token"
  debug:
    msg: "{{ keycloak_token.json.access_token }}"

- name: "Create new user name:"
  uri:
    url: "{{ keycloak_base_url }}/auth/admin/realms/
        {{ keycloak_realm_name }}/users"
    method: POST
    status_code:
      - 201
    headers:
      Content-type: "application/json"
      Accept: "application/json"
      Authorization: "Bearer {{ keycloak_token.json.access_token }}"
    body_format: "json"
    body:
      firstName: "{{ user_firstname }}"
      lastName: "{{ user_lastname }}"
      email: "{{ user_email }}"
      enabled: "{{ user_enabled }}"
      username: "{{ user_username }}"
  register: keycloak_user_create

- name: "Debug var create new user"
  debug:
    msg: "{{ keycloak_user_create }}"

vars/main.yml
keycloak_admin_user: "changeme"
keycloak_admin_pass: "changeme"
keycloak_base_url: "https://keycloak.server"
keycloak_realm_name: "myrealm"
user_firstname: "changeme"
user_lastname: "changeme"
```

```
user_email: "changeme@changeme.com"
user_enabled: "true or false"
user_username: "changeme"
```

```
keycloak_create_user.yml
- hosts: localhost
  connection: local
  gather_facts: false
  roles:
    - create_user
```

G. 管理情報のデータ構造の例

管理情報の定義の例

```
.json
{
    "metrics":{
        "container":"example_val",
        "cpu":"example_val",
        "diskio":"example_val",
        "healthcheck":"example_val",
        "info":"example_val",
        "memory":"example_val",
        "network":"example_val",
    },
    "logs":"example_val"
    "period":"10s"
}
```

管理情報のデータの例

```
.json
{
    "container":"example_val",
    "cpu":"example_val",
```

```
    "diskio":"example_val",
    "healthcheck":"example_val",
    "info":"example_val",
    "memory":"example_val",
    "network":"example_val",
}
.log
log:timestamp=2023-01-11T12:34:56.789Z,module=idp,data:cpu=example_val,
data:mem=example_val,data:disk=example_val,healthcheck=example_val,
info=example_val,memory=example_val,network=example_val"
```

管理情報の出力の例

管理情報の定義と管理情報のデータを key でマージして出力する管理情報を生成、要素の取捨選択も可能 (この例では、healthcheck、info を削除)

```
.json
{
    "container":"example_val",
    "cpu":"example_val",
    "diskio":"example_val",
    "memory":"example_val",
    "network":"example_val",
}
```

H. テンプレートおよびファンクションのデータセットの例

テンプレートの例

```
# function で、function.yaml に定義されている操作を呼び出す
tasks.yaml
- tasks
  - name : ”create new user”
    function : create_user
    env :
```

```
      username : "test"
      password : "p@ssw0rd"

  - name : "delete current user"
    function : delete_user
    env :
      username : "test"

# process に直接操作を記載可能、function に定義することで簡略化が可能
  - name : "get user info"
    process : "GET {{url}}/auth/admin/realms/{{realm}}/users?
              username={{username}}"
    body :
      username : "test"
```

ファンクションの例

```
# process に直接操作を記載し、function を定義する
function.yaml
- functions
  - name : "create_user"
    process : "POST {{url}}/auth/admin/realms/{{realm}}/users"
    body :
      username : "test"
      password : "p@ssw0rd"

- functions
  - name : "delete_user"
    process :
      - "GET {{url}}/auth/admin/realms/{{realm}}/users?
        username={{username}}" > user_info
      - "DELETE {{url}}/auth/admin/realms/{{ keycloak_realm_name }}
        /users/{{user_info}}"
    body :
      username : "test"
```