Architecture Guide

01 March 2021

Metadata

Review

Accountable Director	CTO
Policy Author	CTO
Date Approved	Sep 2017
Date Last Reviewed	
Date of Next Review	

Document history

Version	Date	Author	Description
1	Sep 2017	CTO	Intitial Version
2	Jul 2019	CTO	Document Deprecated
3	May 2020	CTO	Reintroduced document, added version metadata and made major updates
4	Sep 2020	CTO	Further context on scalability
5	Mar 2021	CTO	Updates on deployment model

Contents

1 Infrastructure

1.1 Overview

1.1.1 Deployment Model

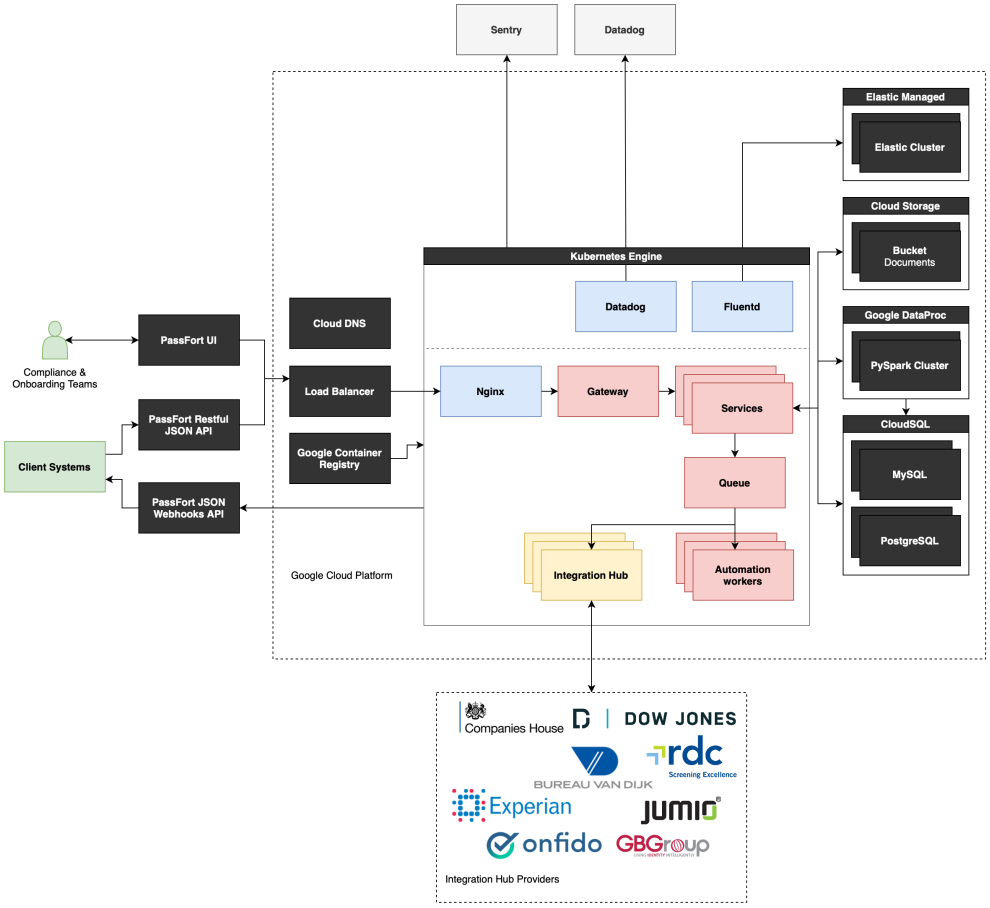
PassFort can be delivered in two ways:

- **SaaS:** All clients run on a shared infrastructure. Please see below for a list of locations.
- **Managed on-premise:** You have your own dedicated infrastructure, although PassFort runs and manages the software. Please contact us for more details.

Location	Provider	Backup Location	Premium Pricing?
Belgium	Google Cloud Platform	Other GCP EU locations	No
UAE	Microsoft Azure	UAE	Yes
Elsewhere	*Contact Us*		

1.1.2 High Level Architecture

A high level diagram of the system architecture follows, along with written descriptions.



PassFort runs as a micro-services application on top of Kubernetes. It is backed by three types of data store:

- **Databases:** We use both MySQL and PostgreSQL databases. These are run in high availability mode, with a hot standby server.
- **File storage:** We store files in Cloud Native Storage, and these are encrypted (at rest) within the application level. Where available, these are versioned, and double encrypted using the cloud provider's native encryption.
- **Search:** We maintain a managed elastic search cluster to power the search features of the application.

We have a data processing pipeline built in PySpark.

We use Hashicorp Vault for key management.

We utilise three products for monitoring PassFort:

- **Prometheus/Grafana/Loki** is used for logging, metric monitoring & APM. We use on-premise versions of these services.

- **Elastic** is used for log management and search.
- **Sentry** is used for error reporting and investigation.

We have CircleCI for continuous integration and storage. Our codebase is managed in GitHub.

All access to PassFort is over HTTPs. We enforce TLS 1.2 and above, in line with industry standards. We integrate with LetsEncrypt to generate certificates.

1.2 Cloud Security

Our SaaS services are deployed on top of mainstream cloud providers. They are responsible for much of the low level security, including physical data-centre security and network security. We also occasionally use of some of their managed offerings.

You can find more infomration about their security policies below.

Google Cloud Platform You can find more information about GCP's approach to security at https://cloud.google.com/security.

Microsoft Azure You can find more information about Azure's approach to security at https://docs.microsoft.com/en-us/azure/security/fundamentals/.

1.3 Prisma Cloud

We have purchased a commercial cloud security solution called Prisma Cloud. This includes:

- Container vulnerability scanning, both pre-runtime and at run-time.
- Web application firewall (WAF)
- Infrastructure configuration compliance & audit – ensuring that our cloud setups do not deviate from best industry practices

1.4 Audit Logging

All events taken within our infrastructure are audit logged by our infrastructure providers. These audit logs are monitored by Prisma Cloud, and alerts are generated when configuration is changed, or falls outside of specified parameters.

2 Application

2.1 Overview

PassFort services are exclusively accessed over a single Restful JSON API, with JSON webhooks used to communicate to client systems. Our services are written in a combination of Python and Rust. Our services are configured as code via terraform and kubernetes.

We are migrating core services to Rust to leverage its strong type safety, and its ability to obviate certain classes of error. Find out more at Why choose Rust?.

Our front end web application (the PassFort Portal) is a single page application written in Javascript (on the React framework). This consumes the same API that is made available to clients. We use flow to improve type safety.

2.2 Access Management

2.2.1 Authentication

Clients can authenticate with PassFort in a number of ways:

- **Username and password:** Passwords are salted and then hashed using SHA512. After initial sign in we utilise sessions to maintain authenticated. This authentication method can be augmented with a second factor (2FA) – we currently support SMS.
- **Single Sign On (SSO):** PassFort is integrated with Okta which allows us to leverage most industry leading SSO providers. As above, we use sessions after the initial authentication. In many cases we can synchronise teams into PassFort from the upstream SSO provider.
- **API Key:** System-system communication requires a 192bit API Key. All accounts have a master API key (which can be regenerated), but secondary keys can be instantiated with scoped permissions. This mechanism requires the token to be passed as a HTTP header with every request.

2.2.2 Authorisation

PassFort supports a powerful permission system across its infrastructure.

These permissions can be configured in the UI, and typically allow for “read”, “write” or “no” access across the different areas of the PassFort product. Permissions can also be scoped around specific

products.

Permissions are grouped into named roles, which can again be configured through our UI. These can finally be attached to either a “team” (which grants the role to all team members), or to individual users.

The permissions system is implemented internally via JWTs (JSON web tokens). Our gateway attaches a JWT describing the permission to the request. These JWTs are verified by a common library which is used across our micro-services.

This authorisation system is the primary way through which client data is segregated. It has been validated through a 3rd party security audit.

2.3 Data

2.3.1 Data Storage

PassFort stores personal data across two databases - MySQL and PostgreSQL. We utilise these two technologies while we migrate from MySQL to PostgreSQL. In our SaaS offerings, these databases are managed by the cloud provieders through their native SQL products (e.g. GCP’s CloudSQL). They run in high availability mode, with hot standbys. All data is encrypted at rest and in transit.

We also collect files from some customers. These are encrypted at the application level (with per-institution keys) using AES256. Where possible, we double-encrypt these files by using our infrastructure provider’s native encryption-at-rest tooling, as an additional layer of security. These are then backed up in a mirrored storage bucket in another region. Access control to these buckets is managed by our terraform configuration.

2.3.2 Data Backups

A detailed discussion of our backup strategy is included in our Disaster Recovery Policy.

2.3.3 Deletion

PassFort offers three forms of data deletion:

- **Archival:** this hides a profile from the UI, but maintains all data within the profile. This can be actioned through the API or UI (with a specific set of permissions).

- **Full profile deletion:** this deletes a profile & its data fully. We will track the deletion within the institution level audit log. This allows our clients to meet their obligations as data controllers under GDPR. This can be actioned through the API or UI (with a specific set of permissions).
- **Full institution deletion:** this deletes an institution and all associated data (including the institution audit log). This would typically be actioned after off-boarding a client, within 30 days. This cannot be instructed through the API, and should be actioned through your customer success manager.

2.4 Monitoring & Logging

2.4.1 Audit Logging

All actions within the platform are logged within our application-level audit service. This audit service logs actions taken by:

- Users
- Client applications (through API Keys)
- PassFort's smart policy automation layer

We track audit items at two levels:

- **Profile level:** all events which affect a specific client profile are tracked within that profile's audit trail.
- **Institution level:** events not tied to a specific profile (e.g. smart policy configuration changes; profile deletions) are tracked across an institution-specific audit log.

2.4.2 Application Logging

All services log data into a loki cluster, which lives alongside our application. As policy, logs do not contain PII - we default to logging UUIDs. All logs contain basic audit information, and we maintain them in perpetuity (from Mar 2021; previously we had a 3mo retention window).

2.4.3 Metric Monitoring & Alerting

We use Prometheus and Grafana for listening to metrics and alerting on those metrics. Alerts are filtered into several channels:

- **Out of hours, high priority, alerts:** alerts which indicate a significant event and must be responded to out of hours by our on duty out-of-hours support team.
- **In hours, high priority, alerts:** alerts which indicate a significant event but do not have substantial time pressure. These alerts are reviewed by our in hours support team on the next working day.
- **Pre-promotion alerts:** alerts which are being evaluated for noise before promotion into one of the above channels.
- **Demoted alerts:** alerts which are have been too noisy to be effective and are demoted pending improvement.

2.4.4 Vulnerabilities

We have a 0-critical and 0-high vulnerability policy, and review all others. We have a 14 day SLA-to-resolution from the moment a resolution is published.

We monitor for vulnerabilities using:

- **GitHub monitoring:** this monitors code dependencies for vulnerabilities.
- **Prisma:** this monitors our containers (both in the registry and those running in our cluster).

2.5 Scalability

2.5.1 Context

Creation vs updating profiles PassFort recomputes a smart policy from scratch every time a profile is created or updated (whether the update is from a check provider or from a customer). PassFort looks to optimise this recomputation (e.g. by caching results), but from a load-testing perspective it is safe to assume that creates have similar performance.

We therefore load test profile creation, as in the vast majority of cases this will be the more expensive of the two operations.

Bottlenecks PassFort's architecture has several places which could theoretically cause a bottleneck. The highest risk areas are:

- **Database performance:** PassFort's customers require strong guarantees around consistency and durability of their data. This means that all operations which alter a profile must be committed to a centralised database before returning.

- **Workers:** the vast majority of work in PassFort's systems is done asynchronously via a message queue and a set of worker processes. If there are not enough of these to service load, then KPIs that our customer's rely on (e.g. time to decision) could start to increase.
- **Application locking behaviour:** Our application handles all database locking at the profile level – enforcing serialised access. This means that we can expect success handling a higher number of profiles via horizontal scaling initiatives.
- **Application services & workers:** These are stateless, and therefore can be scaled horizontally. Given there are hosted on kubernetes within the cloud, we are able to scale these within minutes.

Load from other clients PassFort's SAAS offering involves clients using shared infrastructure. To mitigate the risk of a traffic spike from one client impacting others, we have client-specific queues within our message broker. Workers operate on a round-robin approach to guarantee quality of service to all clients.

2.5.2 Internal Tests

Setup PassFort was run in a small production environment:

- c. 50 workers
- DB with 2 cores & 7.5GB RAM
- We disabled all system autoscaling.

We configured an account with a complex script (based off of real customer configurations), including all main modules:

- Risk
- Branching policy flow with 20 branches
- Waterfalled checks
- Providers – although these were run in demo mode - i.e. we did not make calls to third party providers

We ran a script which created profiles in parallel at a configurable rate. We measured a number of KPIs off the back off of this script, including API success rate, latency, profile time-to-decision. We ran this script in stepped increments of 1000 profiles per hour (i.e. we ran at 1000p/h, 2000 p/h, ...).

Results We successfully tested the system to 30,000 profiles per hour (equivalent to 720,000 profiles per day). At all levels we saw good API performance.

Between 0 and 5,000 profiles per hour we saw profile time-to-decision stay flat, as we saturated our workers.

Between 5,000 and 20,000 profiles per hour we saw time to decision increase predictably and linearly to a total of 3x baseline. Given the stable results, we believe this was due to worker saturation (which occurred as we had disabled autoscaling of our systems).

Between 20,000 and 30,000 profiles per hour we saw time to decision fluctuate between 3x and 10x baseline. Reviewing our monitoring we saw that we started to hit the limits of our database.

Plans To push performance beyond the above numbers, we have a number of options:

Scaling the numbers of workers (this is already in place, but was disabled for the performance test).

Improving specifications of our database (we typically run production on a larger database to provide 10x headroom).

Improvements to how we pipeline automations to minimise database contention (work is beginning in Q4)

Horizontal sharding of the database (not currently forecast)

3 Operations

3.1 Access

Access to production systems is restricted.

Three users are designated administrators have complete access to production systems. This access is only used in exceptional circumstances.

The vast majority of changes are made via configuration-as-code. These changes go through our standard SDLC, where changes are reviewed and approved by peers and senior engineering management. This is then deployed into our staging and production environments by our CI/CD systems.

Our infrastructure is accessed via a VPN, which requires 2 factor access (hardware token).

3.2 Environments

PassFort maintains a production and a staging environment. These are functionally identical, with similar access restrictions. The staging environment runs on a copy of production data (made at least monthly).

3.3 SDLC

3.3.1 Design

The first phase of our SDLC involves doing design work on a feature. As part of this we do a risk assessment of the feature. This informs our testing plan and deployment plan. When a feature is high risk, it requires review and sign off from senior engineering stakeholders at the design phase.

3.3.2 Testing

PassFort has an extensive automated test suite. This includes:

- **Unit testing:** testing specific functions within a service.
- **Feature testing:** testing flows within a specific microservice, with other services mocked out.
- **Integration testing:** testing our microservices in aggregate. We exercise this via API calls, headless browser UI tests, along with visual regression tests.

These tests are run via CircleCI as part of development and as part of our deployment process.

3.3.3 Review & Approval

All features are peer reviewed, with at least one review required before a feature is merged into our codebase. We run a secondary risk assessment as part of this review process, identifying any material changes between the design and the eventual implementation.

For features which are identified as high risk, reviews are required from senior engineering stakeholders.

Where possible, we also run static analysis tooling to capture common errors.

3.3.4 Deployment

PassFort services are continuously deployed to – we deploy to them multiple times a day.

All PassFort services are have automated CI/CD.

We release features on Monday to Thursday. We only release hot-fixes out of these hours. If PassFort has to perform maintenance which will result in downtime, we target a maintenance window of 8-9am on Sundays. This will be published ahead of time on https://status.passfort.com.

3.4 Maintenance

PassFort, or our the providers of our managed services, occasionally have to perform maintenance that causes downtime. This typically includes upgrading the versions of our core infrastructure (e.g. our databases). Although we try to avoid these events, they usually happen 1-2 times per year. Recent downtimes have only been for a few minutes.

We have a consistent maintenance window which has been judged across our entire (global) client base's peak operating times. This is Sunday 8-9am UTC. We always provide advanced notice, through https://status.passfort.com.

3.5 Support

PassFort support cafn be accessed via support@passfort.com. For those on our premium support package, there is also a 24/7 phone number for P1 issues.

For more information about SLAs, please refer to your contract.

PassFort has on duty engineering 24/7. We split these into an “in hours” (IH) rotation and an “out of hours” (OOH) rotation:

IH engineers act as a second line of support, for both client issues and internally generated issues (e.g. alerts from our infrastructure monitoring).

OOH engineers are available to fix P1 issues identified by our clients and by our infrastructure monitoring.