

**Collaboard Architecture**

OVERVIEW

|  |  |
| --- | --- |
| Title: | Collaboard Architecture Overview |
| Abstract: |  |
| Status\* | Approved |
| Document Owner: | IBV |
| Document Author: |  |
| \*Draft, In progress, Reviewed, Authorized, Approved, Completed | |

|  |  |  |
| --- | --- | --- |
| **Document Control** | | |
| Document Distribution: | Internal: | All dev Teams |
|  | External: |  |
| Document Authorization: | Authorizer(s): | GP Santopaolo |

|  |  |  |  |
| --- | --- | --- | --- |
| **Change History** | | | |
| Date | Ver | Changed By | Details |
| 26/02/2019 | 1.0 | GP Santopaolo | Initial draft |
| 19/05/2021 | 1.1 | GP Santopaolo | Review and updates |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Index

[Introduction 3](#_Toc72506287)

[Purpose of the document 3](#_Toc72506288)

[Foreword 3](#_Toc72506289)

[CollaBoard architecture – all services 4](#_Toc72506290)

[OpenShift, Kubernetes, and Docker scenario 5](#_Toc72506291)

[Standard Web API request 6](#_Toc72506292)

[Real-time data exchange 6](#_Toc72506293)

[File upload download 6](#_Toc72506294)

[Telemetry data 6](#_Toc72506295)

[Web RTC 6](#_Toc72506296)

[gRPC 6](#_Toc72506297)

[Architecture layer explained 7](#_Toc72506298)

[Azure Scenarios 8](#_Toc72506299)

[Running on Azure using Blob Storage 8](#_Toc72506300)

[Important note when using Blob Storage 9](#_Toc72506301)

[Running on Azure using PaaS services 9](#_Toc72506302)

[Running on Azure without PaaS Services 10](#_Toc72506303)

[Azure multi-region 10](#_Toc72506304)

[Azure multi-zone 11](#_Toc72506305)

[AWS Scenarios 12](#_Toc72506306)

[Running on AWS using CloudFront and S3 12](#_Toc72506307)

[Running on AWS using PaaS services 12](#_Toc72506308)

[AWS without PaaS services 14](#_Toc72506309)

[AWS multi-region and multi-zone 14](#_Toc72506310)

# Introduction

## Purpose of the document

Collaboard, from an architectural point of view, is a highly flexible application. It can run on the Cloud by using only containers, or when requested, it is able to take native cloud technologies and integrate several PaaS services.

Collaboard can run on the Cloud: AWS EKS, Azure AKS, Google GKE. Or on-premises within the customer's boundaries: Kubernetes, OpenShift, or even Docker-compose (for very small POCs) are all supported scenarios where Collaboard can be installed.

This document will describe the various possible scenarios in different environments

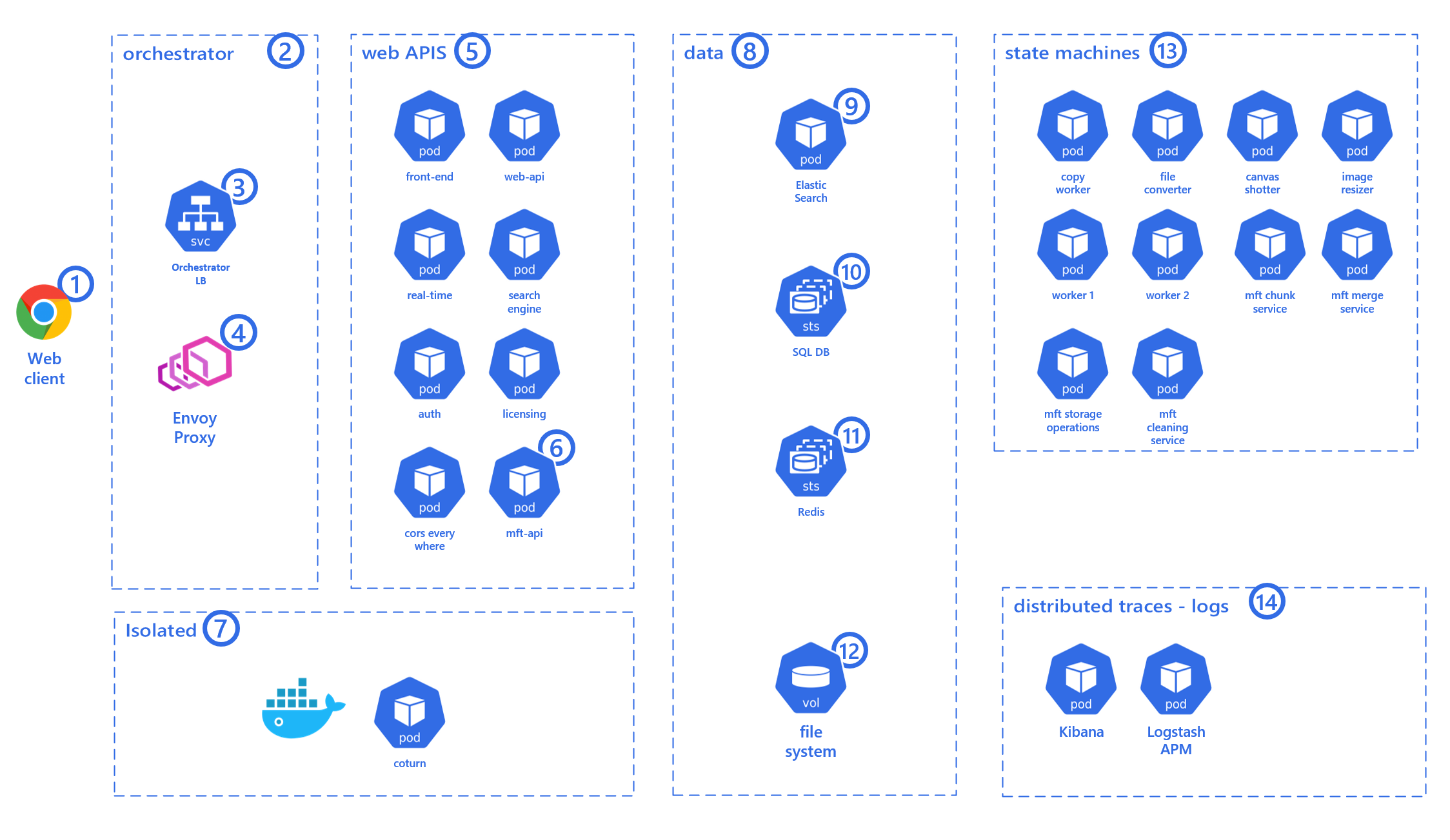
All the connection described in this document are secure (https)

# OpenShift, Kubernetes, and Docker scenario

Our infrastructure consists of several layers (Diagram 1):

* The orchestrator (2) and Web APIs (5) layers are the only ones available for the client.
* The data layer (8) is only accessible internally and contains all our repositories
* The state machines (13) are only accessible internally, and they are responsible for running all our worker processes
* The distributed traces (14) are only accessible internally. Optionally the customer can decide if to expose Kibana publically. Kibana is the web UI showing all the traces and logs.
* The isolated layer is publically visible, but it runs outside the architecture. It runs in an isolated docker-compose instance. In this area, we store all the web servers needed for the clients to expose some optional functionality (mouse movements) correctly that they don’t directly relate to our infrastructure.

The isolation level is also essential to avoid exposing the application to any potential security risk.



*Diagram 1 Collaboard, complete architecture*

When a web client (1) connects to the online whiteboard URL (https://web.collaboard.app on our public environment, it can be any URL), it requests a connection to orchestrator (2) the load balancer (3) together with the proxy (4) decide to which pod (container instance) to route the request.

When a Collaboard client deals with services (5), it can perform several service requests that we can categorize as:

1. Standard Web API request
2. Real-time data exchange (web socket)
3. File upload and download
4. Telemetry data
5. Web RTC
6. gRPC

## Standard Web API request

The client (1) performs a service request to the Web API (5). The business layer process the request and, data are collected or stored in the data repositories (8). The response is sent back to the client (1).

## Real-time data exchange

When the client (1) runs the web application, it is always connected to a Real-Time service (real-time pod in 5) based on WebSockets. Thanks to this approach, all the clients participating in the same projects can send and receive data from each other.

The same data, when needed and when the business layer decides that is appropriate, are also stored in the database (8) or sent to the state machines (13) for offline processing

## File upload download

The client (1) can upload and download files of any type and size. Collaboard can achieve the same result by using IBV’s MFT, Azure Blob Storage or AWS S3 With Cloud Front.

Using native Cloud technology (Blob Storage or S3 with CF), it's effortless to transfer terabytes of data across clients and servers all over the World.

No known technology allows files of any size to be transferred over a network when it comes to on-premises. No matter if it is a small local network or a wide area network. Transfer a file of any given size in a reliable way it is not possible.

That’s why we developed MFT (managed file transfer), our custom technology that allows Collaboard to handle files of any size, no matter how many parallel requests we get.

In diagram 1 above, the application uses our MFT (Managed File Transfer) component to achieve the result of uploading and downloading files of any size from a remote client to the collaboard infrastructure.

## Telemetry data

Each action performed on the client (1) and server are stored on the Elastic Search (7) we have two different Telemetry category:

1. Application logs, including exceptions
2. Performed actions stored in an anonymous form (e.g., upload action, project creation action, and so on)

If the customer decided that we need to store client (1) telemetry data, then the logstash APM service (14) needs to be reachable by the client.

If there isn’t a need to store client telemetry data, then the Logstash APM service can be set to be reached internally only.

The same applies when using Application Insight on the *Running on Azure without PaaS Services* or Running on AWS without PaaS services scenarios.

Any Business Intelligence can later analyze these data.

## Web RTC

The client (1) failed to establish a peer-to-peer connection with other clients; it backs up using the TURN and STUN server (coturn) (7).

## gRPC

The client (1) establishes a direct communication channel, unidirectional from the web APIs (5) layer, to receive notifications.

gRRC will also be used as a bidirectional communication channel for server-to-server communication (internal only).

This functionality will be GA (generally available) in a future release.

## Architecture layer explained

The **orchestrator layer** (2) is the entry point of our application all the TCP traffic coming from clients (1) is analyzed by the orchestrator load balancer (3) and the application proxy (4).

They decide which web APIs(5) will be responsible for serving the particular web request.

The orchestrator LB (3) and proxy (4) are also responsible for scaling up and down all the pods (5-8-13 and 14) based on their internal rules.

When the application runs in the customer’s environment, the customer, with our engineers, can decide the best rule for the environment.

The **web APIs** (5) are responsible for serving the various client requests. They contain part of the business logic of the application. They are responsible for storing data in our data repositories (8), broadcasting messages to clients, and to create queues for the worker services (13).

We store **data** (8) in various repositories:

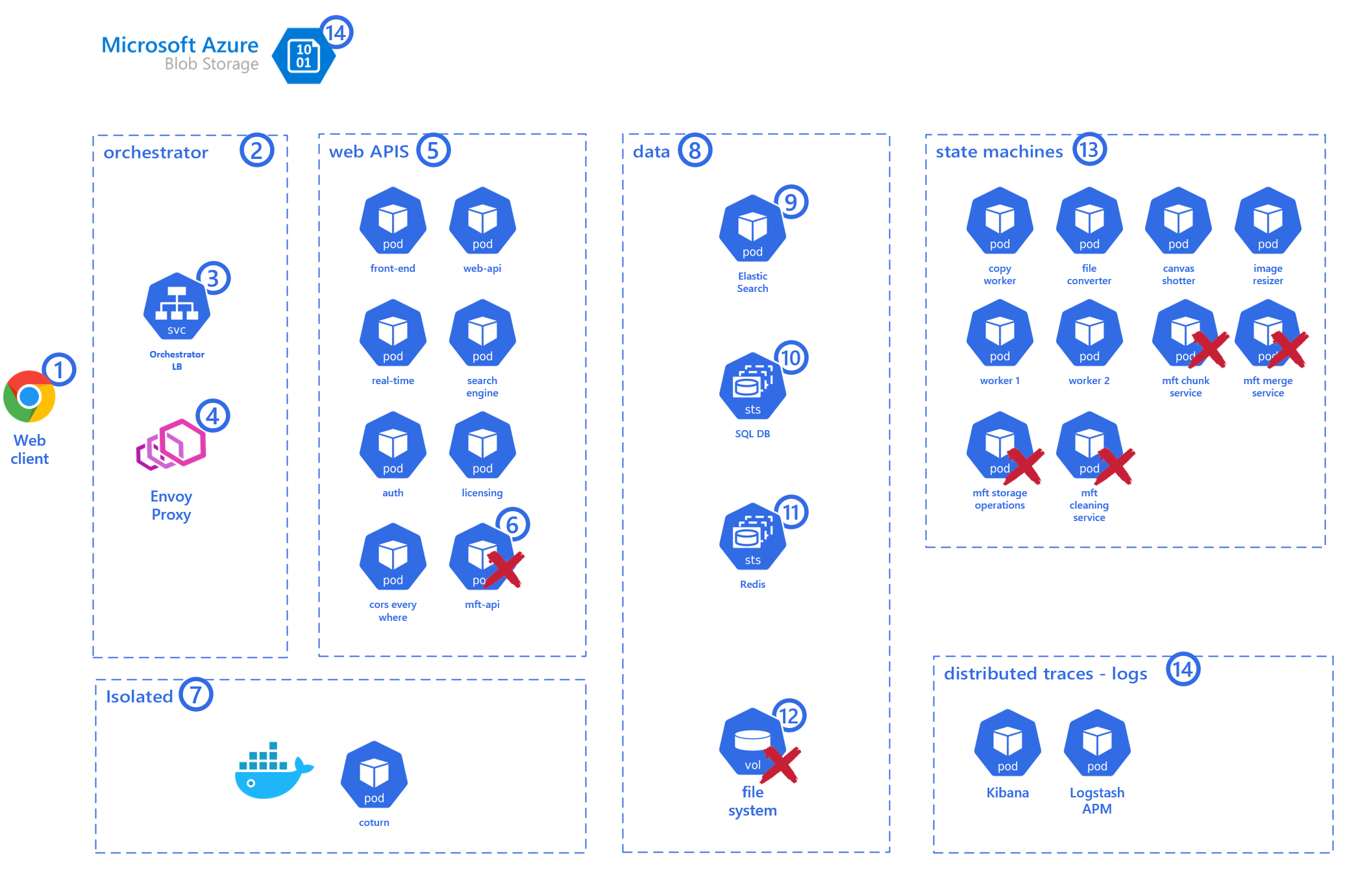
* On SQL Server (10), we store all the information related to canvas, users, and projects
* On Elastic Search, we store all the metadata needed for the client to perform fast end efficient searches
* We use Redis as a backplane for the real-time web socket connections to scale-out. In the future, we will also use it for other features
* We use the file system (12) mounts to store and retrieve files stored in our canvases (projects)

# Azure Scenarios

## Running on Azure AKS using Blob Storage

As already said, Collaboard, from an architectural point of view, is a highly flexible application. It can run on the Cloud by using only containers, or when requested, it can take native cloud technologies and integrate several PaaS services.

An example of Collaboard's flexibility is the use of storage. We all know that each cloud provider has its native offering when it comes to storage. With Azure Blob Storage, it's effortless to transfer terabytes of data across clients and servers all over the World.



*Diagram 2 Collaboard running on Azure using Blob Storage*

In the scenario above, we are running Collaboard on Azure using Blob Storage.

This scenario reduces the maintenance effort for the application because Microsoft will care about all the procedures to keep the storage safe and reliable.

Disaster recovery, fault tolerance, and backup are all functionalities we will get out of the box.

When using Azure Blob Storage in our solution, the Blob Storage will be plugged in, replacing the MFT component.

There will be no need for the mft-api container (6) and the MFT services marked with the red cross in the state machines (13).

There will be no need for a file system (12) mount in the data (8) layer.

### Important note when using Blob Storage

There is a side effect when taking advantage of the native cloud service to work with files. It can be important for some customers to be aware of it.

When Collaboard uses the Blob Storage (14), the client (1) needs to have the proper logic and access permission to access the Azure Blob Storage (14) directly.

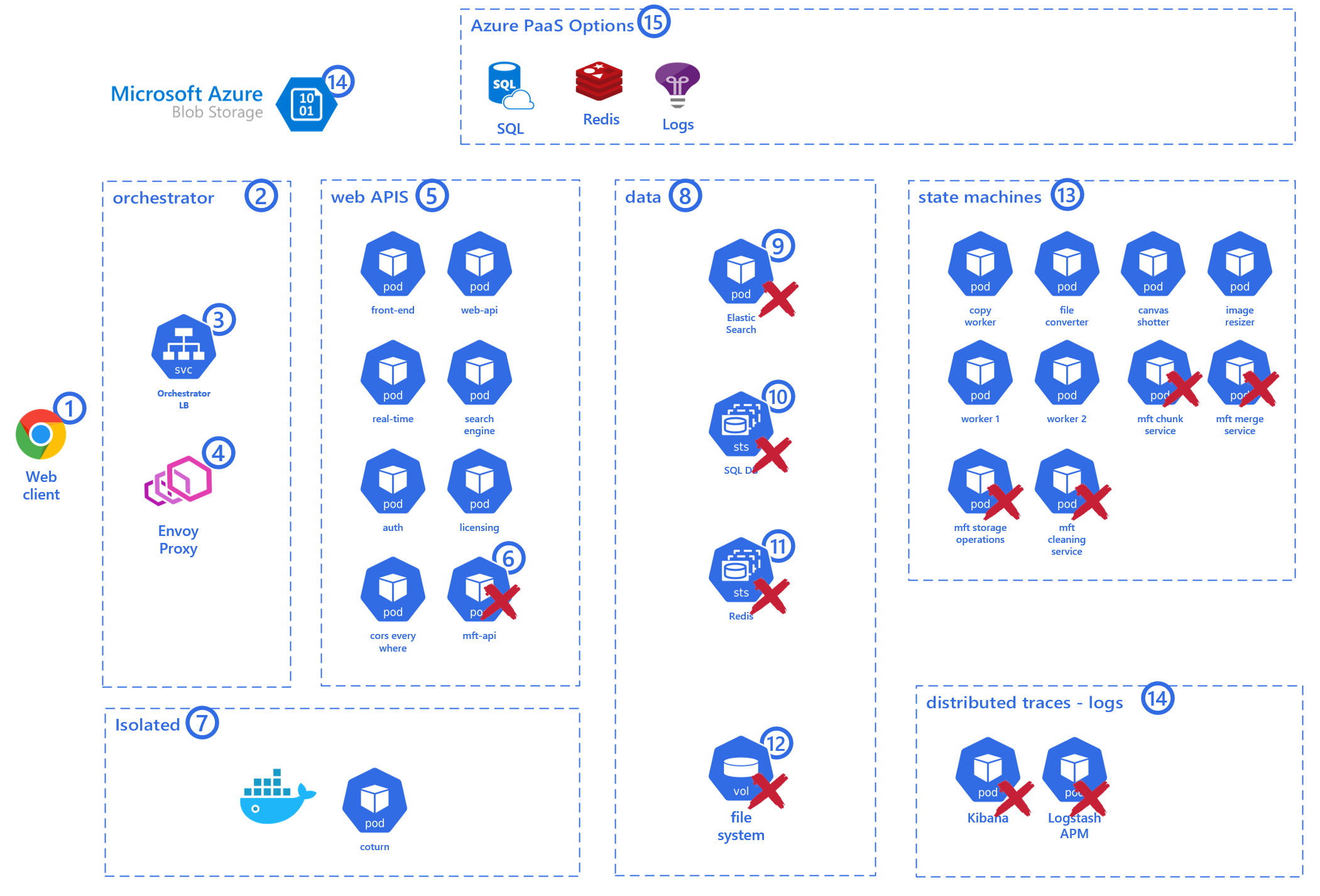
This means that the Azure Blob Storage needs to be publically exposed even though a client needs to provide the right access token to access the resource.

For all the insight about accessing a cloud storage resource with Collaboard, please refer to the *Security&Auth - SAML and Storage – Collaboard* document.

## Running on Azure AKS using PaaS services

When running on the Cloud, with no downside effects, we can take advantage of the following PaaS:

* SQL managed instance
* Redis Cache
* Application Insights (distributes traces and logs)



*Diagram 3 Collaboard running on Azure AKS using PaaS services*

In this scenario, the whole data layer (8) is provided as a service by the Cloud provider.

The Cloud provider also provides logs, distributed traces, and storage as a service.

In this scenario, there will be no need for the containers marked with the red cross because Collaboard will use the comparable service provided by the cloud provider.

The advantages of this scenario are that it dramatically reduces the maintenance effort for the application because the Cloud provider will care about all the procedures to keep the storage, data, and logs safe and reliable.

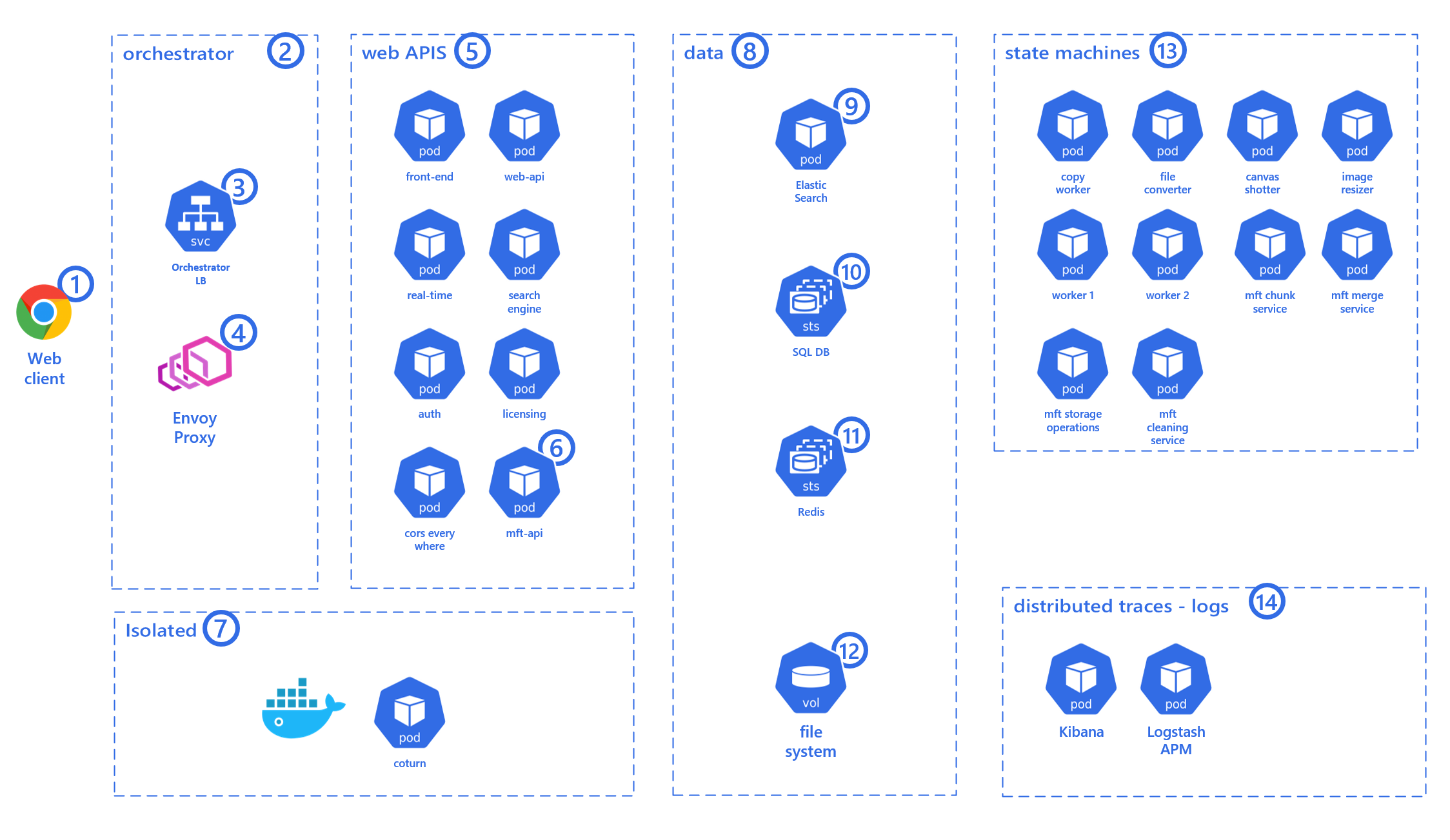
Disaster recovery, fault tolerance, and backup are all functionalities we will get out of the box.

The consideration about the storage is the same explained in the *Important note when using Blob Storage* paragraph.

The customer can choose to use each of the PaaS services described individually

## Running on Azure AKS without PaaS Services

If for any reason, the customer does not want or cannot use any of the PaaS services offered by the Cloud vendor, we can run the application on Azure AKS by using containers only.



*Diagram 4 Collaboard running on Azure AKS without PaaS services*

It is important to note, though, that the customer will be in charge of the procedures and effort needed to keep Collaboard’s data safe and reliable.

The customer will be responsible for the proper setup to provide disaster recovery, fault tolerance, and backup for both the file system and the data (Elastic, SQL Server, and Redis).

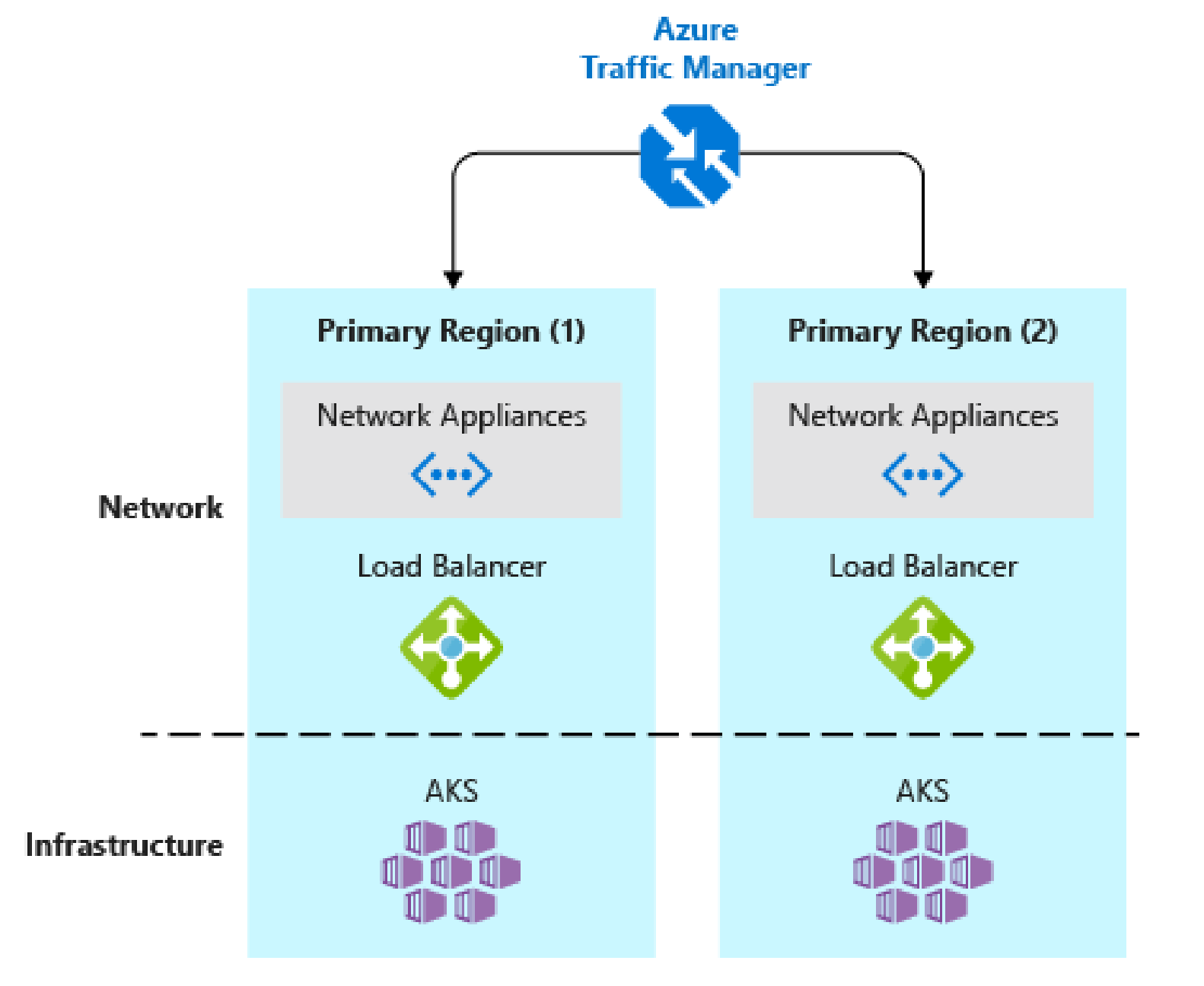
When running on Azure without Blob Storage, Collaboard uses Azure Files for file system (12) exposed internally only. The client (1) will be able to upload and download files through the mft-api (6)

## Running on Azure AKS multi-region

Suppose there is a need to have the application available from different geo locations, like our production environment https://web.collaboard.app. In that case, we allow users from all over the World to connect to the same canvas.

There might also be a need for business continuity or high availability in case of disaster recovery.

In those case, we offer the opportunity to deploy the application on



*Diagram 5 Collaboard running on Azure AKS multi-region*

If there is such a need, we can provide the application deployed on multiple regions.

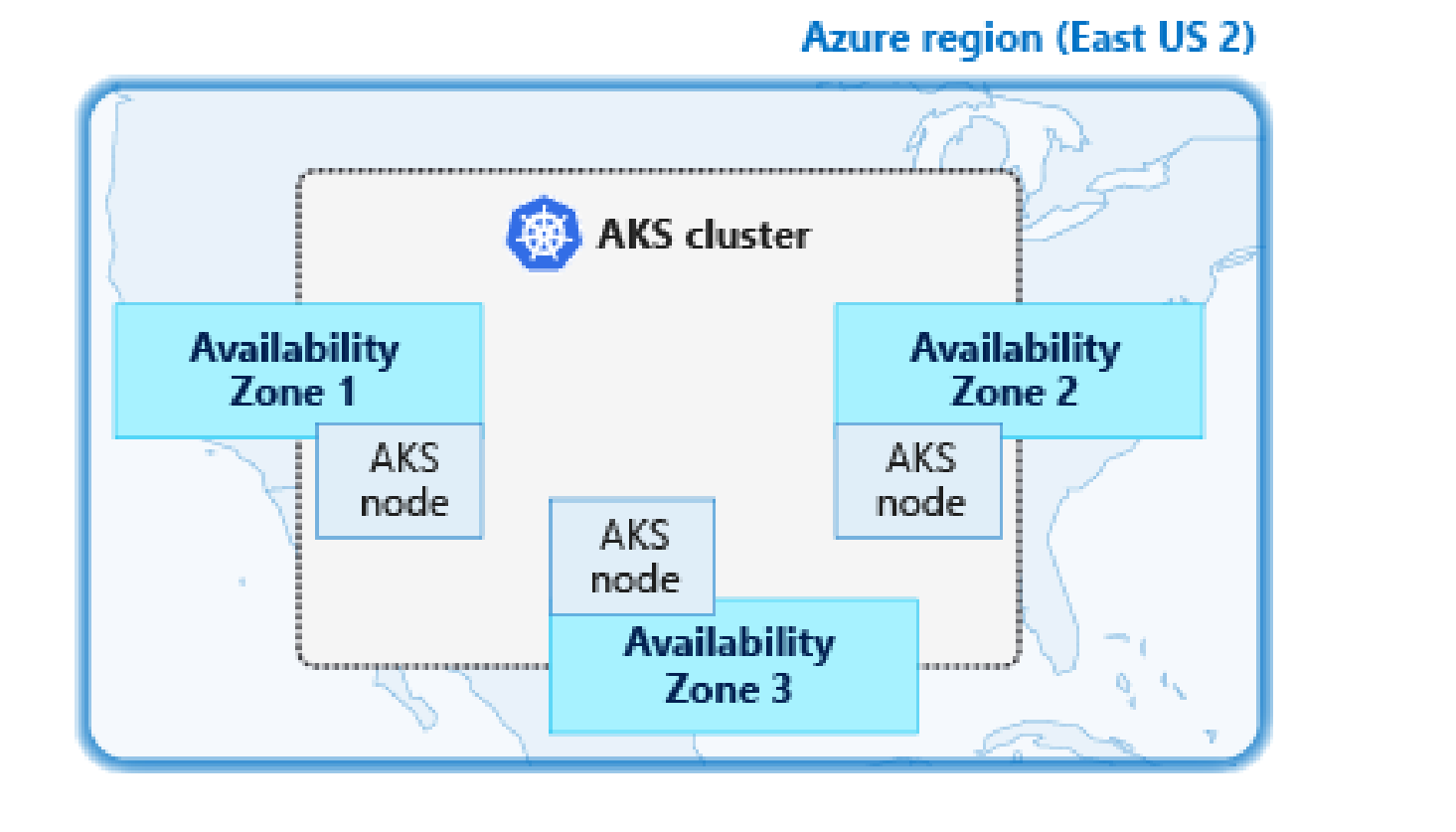
In case interested in this scenario, there is an excellent article on Microsoft docs “Best practices for business continuity and disaster recovery in Azure Kubernetes Service (AKS)”.

<https://docs.microsoft.com/en-us/azure/aks/operator-best-practices-multi-region>

The theory explained in the article applies to Azure AKS and Amazon AWS EKS, we strongly suggest having a read.

## Running on Azure multi-zone

In case of for need for business continuity or high availability in the same region.



*Diagram 6 Collaboard running on Azure AKS single-region, multi-zones*

Within a region, we can have multiple availability zones. You can have one AKS cluster across different zones.

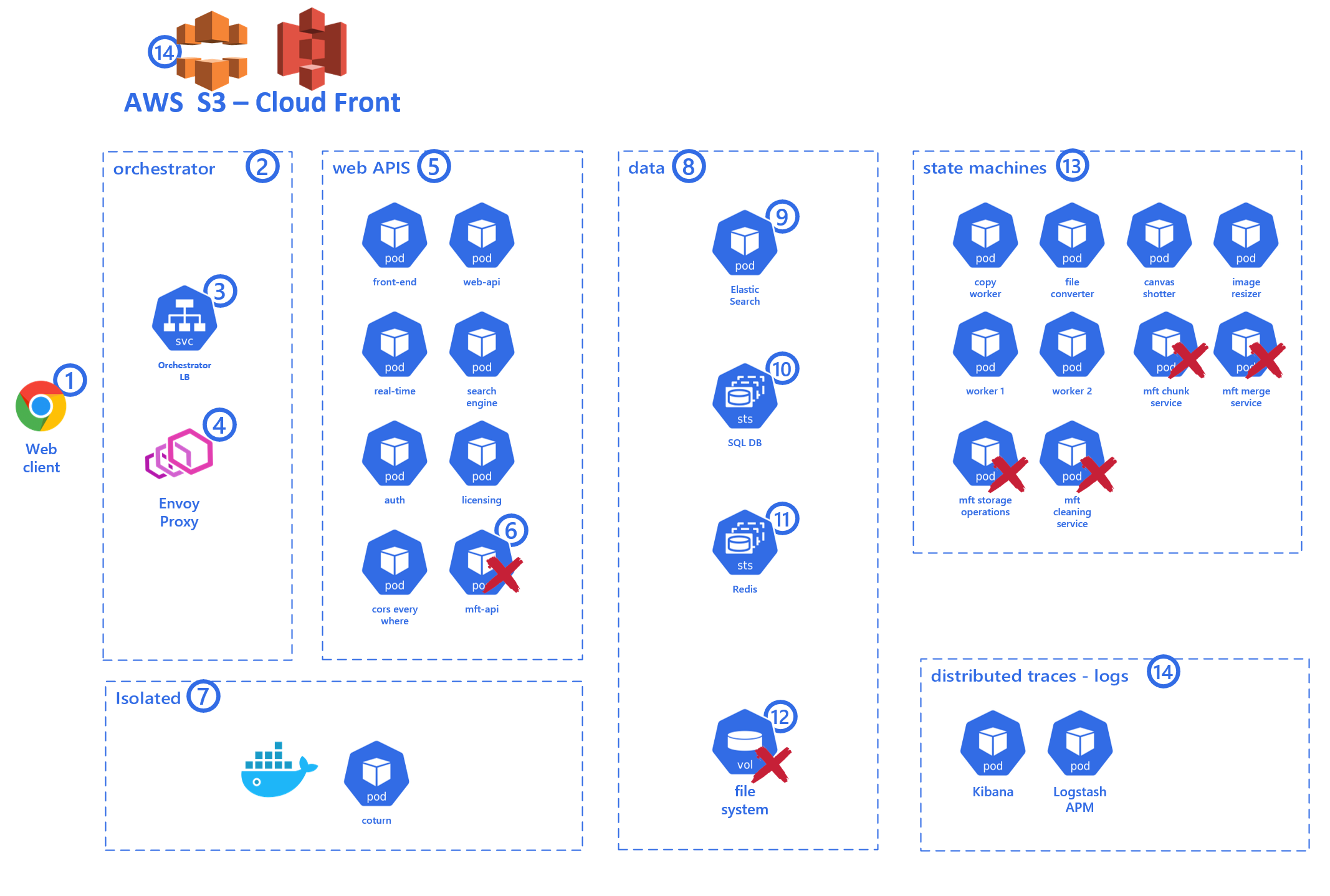
In Azure Gov, only region "US Gov Virginia

# AWS EKS Scenarios

## Running on AWS EKS using CloudFront and S3

Exactly as happens when running on Azure using Blob Storage, Collaboard can quickly run on AWS S3 with Could Front.

With AWS S3 and Could Front, it's effortless to transfer terabytes of data across clients and servers all over the World.



*Diagram 7 Collaboard running on AWS EKS using CloudFront and S3*

In the scenario above, we are running Collaboard on AWS using S3 with Cloud Front.

This scenario reduces the maintenance effort for the application because Amazon will care about all the procedures to keep the storage safe and reliable.

Disaster recovery, fault tolerance, and backup are all functionalities we will get out of the box.

When using AWS S3 with Cloud Front in our solution, the AWS S3 with Cloud Front will be plugged in, replacing the MFT component.

There will be no need for the mft-api container (6) and the MFT services marked with the red cross in the state machines (13).

There will be no need for a file system (12) mount in the data (8) layer.

### Important note when using AWS S3 with Cloud Front

There is a side effect when taking advantage of the native cloud service to work with files. It can be important for some customers to be aware of it.

When Collaboard uses the AWS S3 with Cloud Front (14), the client (1) needs to have the proper logic and access permission to access the S3 storage via Cloud Front (14) directly.

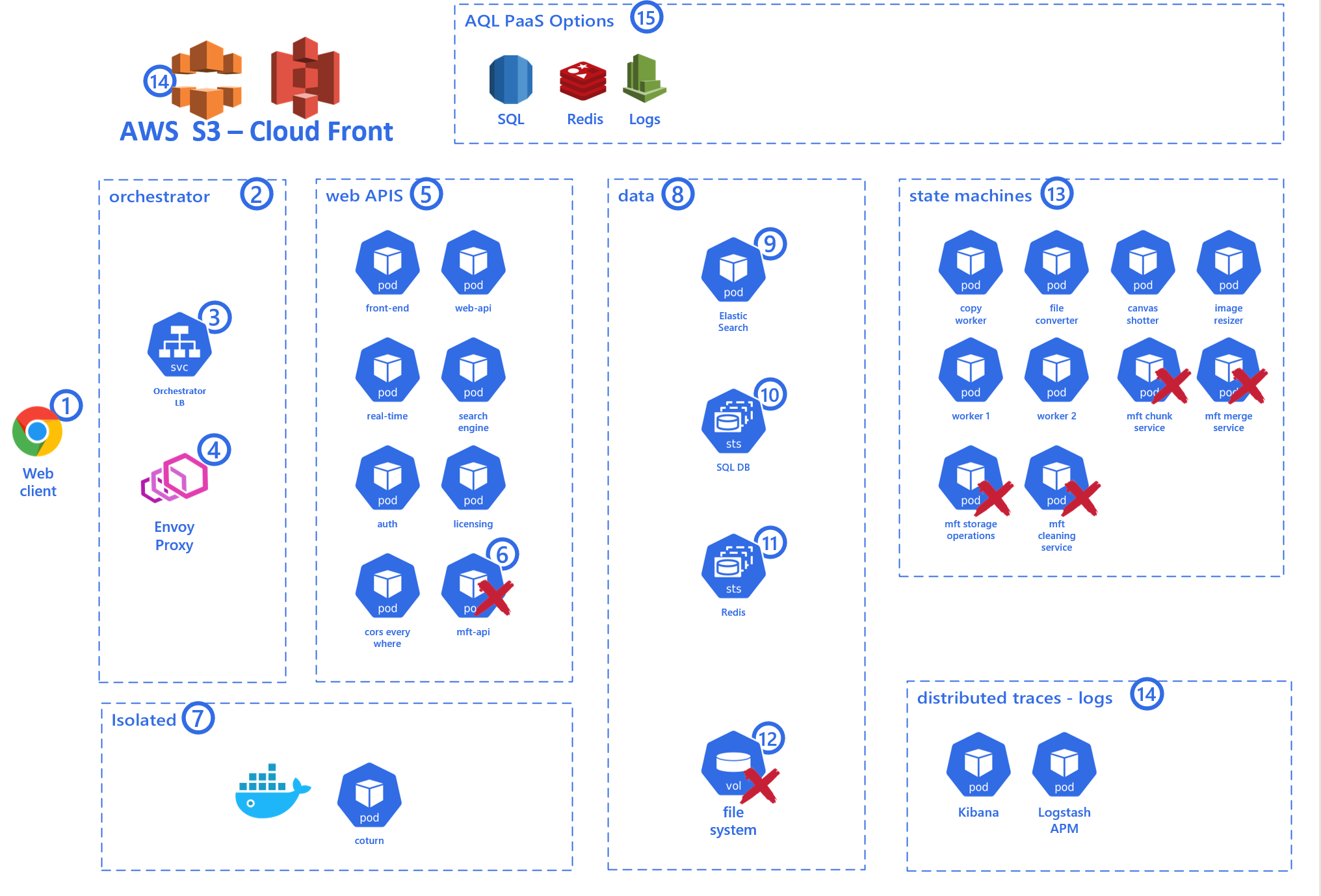
This means that the S3 Storage needs to be publically exposed via Cloud Front, even though a client needs to provide the right access token to access the resource.

For all the insight about accessing a cloud storage resource with Collaboard, please refer to the *Security&Auth - SAML and Storage – Collaboard* document.

## Running on AWS EKS using PaaS services

When running on the Cloud, with no downside effects, we can take advantage of the following PaaS:

* Amazon RDS for SQL Server
* Redis Cache
* Cloud Watch (distributes traces and logs)



*Diagram 8* *Collaboard running on AWS EKS using PaaS services*

In this scenario, the whole data layer (8) is provided as a service by the Cloud provider.

The Cloud provider also provides logs, distributed traces, and storage as a service.

In this scenario, there will be no need for the containers marked with the red cross because Collaboard will use the comparable service provided by the cloud provider.

The advantages of this scenario are that it dramatically reduces the maintenance effort for the application because the Cloud provider will care about all the procedures to keep the storage, data, and logs safe and reliable.

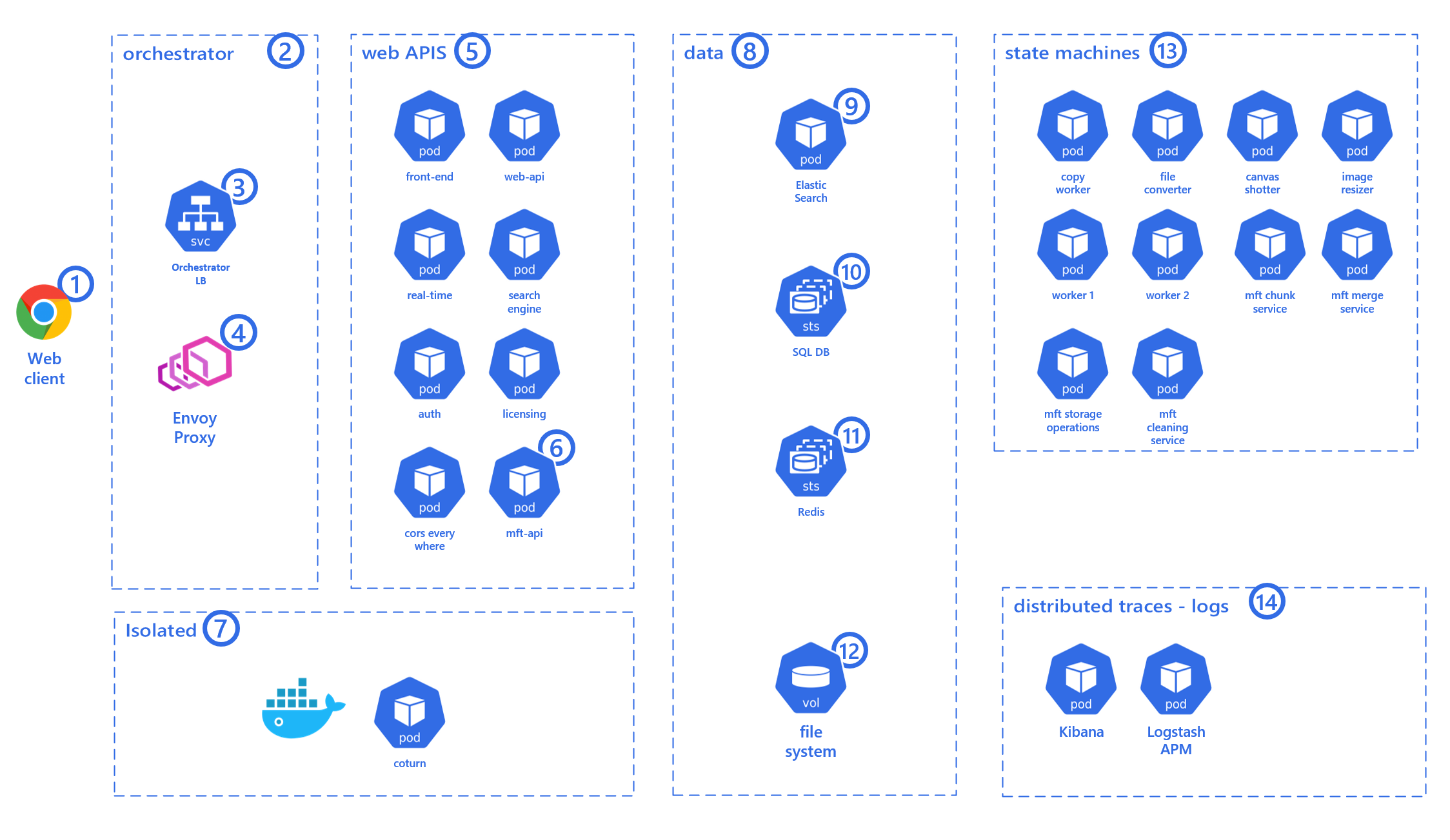
Disaster recovery, fault tolerance, and backup are all functionalities we will get out of the box.

The consideration about the storage is the same explained in the *Important note when using AWS S3 with Cloud Front* paragraph.

The customer can choose to use each of the PaaS services described individually

## Running on AWS EKS without PaaS services

If for any reason, the customer does not want or cannot use any of the PaaS services offered by the Cloud vendor, we can run the application on AWS EKS by using containers only.



*Diagram 9 Collaboard running on AWS EKS without using PaaS services*

It is important to note, though, that the customer will be in charge of the procedures and effort needed to keep Collaboard’s data safe and reliable.

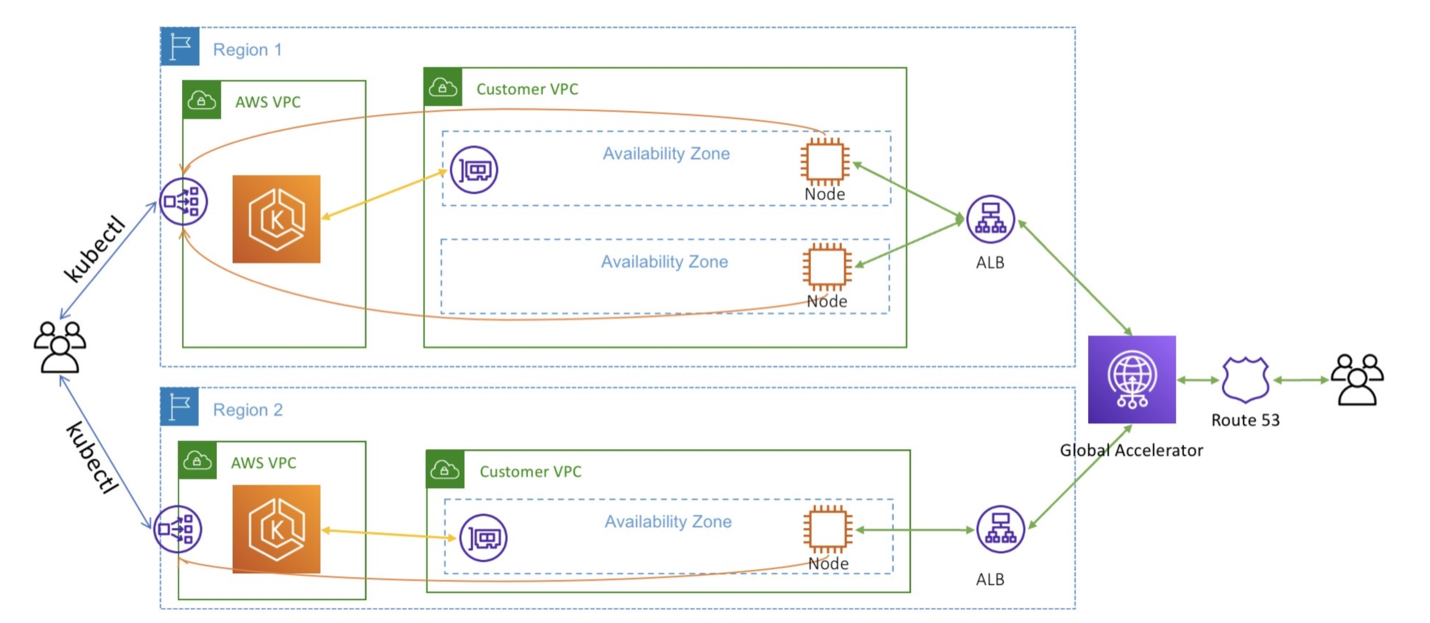
The customer will be responsible for the proper setup to provide disaster recovery, fault tolerance, and backup for both the file system and the data (Elastic, SQL Server, and Redis).

When running on AWS EKS without S3 with Cluod Front, Collaboard uses EFS (Amazon Elastic File System) for file system (12) exposed internally only. The client (1) will be able to upload and download files through the mft-api (6)

## Running on AWS EKS multi-region and multi-zone

In AWS, the recommended way to run highly available Kubernetes clusters is using Amazon Elastic Kubernetes Service (EKS) with worker nodes spread across three or more Availability Zones.

For globally available applications, it is recommended to run separate clusters in different Regions with multi-zone worker nodes. This is very similar to AKS :



*Diagram 10 Collaboard running on AWS EKS multi-region with multiple availability zones*

For having multi-region, we need an EKS cluster in each region and use a Global Accelerator to access the applications running in the corresponding EKS cluster