Master’s Degree Thesis

Kubernetes Cluster On-premises Infrastructure for Stateful Applications

Supervisors

Prof. Guido MARCHETTO

Prof. Jaloliddin YUSUPOV

Candidate

Bulat DAVLYATSHIN

December 2022

Summary

Two events from our past has driven first the technology towards the future, then the world towards the technology. These are the invention of Cloud and the Covid-19 respectively. Nothing made us live in a remote reality as the latter one. Today quite a few of our daily experiences can be handled remote and online. From banking services to education and telemedicine - life will never be the same.

To this prospect, we are facing a high load on our servers, which is increasing sometimes in an exponential manner. Yes, we have nice applications that make our daily routine better, but to keep them working stable we have to ensure that they can always handle the traffic they're enforced to.

Do we have a solution for that? - Yes, we have cloud, we have containerization for application high mobility, we have orchestrating tools like Kubernetes, and all of them do a great job, but as we know, there is no silver bullet. We cannot expand one solution to all of the cases, and one of the cases is studied in this work.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Thesis Objective

Having a legacy infrastructure with legacy source code, that actually perform well, but requiring vertical scaling on high loads, taking into account that load behaves non-uniformly, we took a challenge to shift the paradigm towards horizontal scaling. Also we decided to somehow automate application management (orchestrating) process and faced two different, and in some sense opposite approaches: **Cloud vs. On-Premise**

As long as the University has its own servers, but not a cloud data center yet, we are left with the **On-Premise approach**.

The next issue we deal with is that we work on an application that is stateful. That makes a bit difficult to not loosing the state of the request, otherwise the user experience would be ruined. We have to be coutious in assinging volumes, as **StatefulSets** specifications suggest to use **PersistentVolume** for saving the state even if the pod will go down, after the restart it will catch up where it left off.

We dig deeper, and we see that if we have 2 replicas of the pods that manages its own **PersistentVolume** then we do not have the right user experience, because the same user's requests comes to these two pods interchangably, but one pods info do not intersects with the other's. So, data synchronization comes to a place and different topologies like Master-Slave and Cluster Topologies.

Taking this analysis into account we set the following goals:

1. Install Kubernetes Cluster as a sandbox
2. Study the BigBlueButton application in its legacy architecture and gain some insights for the further conclusions
3. Design a brand new Kubernetes-friendly architecture for BigBlueButton
4. Refactoring the application
5. Adding monitoring tools
6. Design different scenarios for testing and debug
7. Prepare for production

1.2 Thesis Description

This thesis work is structured as follows:

- **Chapter 1 - Introduction**: This chapter, as the name suggests, is an introduction which reveals the objective of the actual work, what ideas led to start this work and how they transform along the way.
- **Chapter 2 - Background**: This chapter makes us familiar with the technology used in this work. Of course in a quite brief manner. We will cover what is Cloud and how it differs from On-Premises solutions, what is a state in an application, and what is Kubernetes.
- **Chapter 3 - Infrastructure Setup**: In this chapter we will be defining the requrements for our future application infrastructure and installing a Kubernetes Cluster on our Virtual Machines.

- **Chapter 4 - Stateful Application on Kubernetes** Given that we successfully installed the Kubernetes Cluster in the previuos chapter, we will try to deploy a stateful application on it. We will see in practice how differently wee deal with Deployments and StatefulSets.
- **Chapter 5 - Conclusions and Future Work** Here we discuss the results we achieved and what has left for the future.

Chapter 2

Background

2.1 Kubernetes

Kubernetes is an open source tool to manage and orchestrate containerized applications. It provides such a powerful set of features as zero downtime deployment, health check mechanism, pod liveness handling, graceful shutdown and many others.

"Kubernetes, or k8s, is an open source platform that automates Linux container operations. It eliminates many of the manual processes involved in deploying and scaling containerized applications," and continues "In other words, you can cluster together groups of hosts running Linux containers, and Kubernetes helps you easily and efficiently manage those clusters." - well known definition by Gordon Haff, Red Hat technology evangelist, in his book, "From Pots and Vats to Programs and Apps" [1]

Kubernetes was originally developed and designed by engineers at Google. Google was one of the early contributors to Linux container technology and has talked publicly about how everything at Google runs in containers. (This is the technology behind Google's cloud services.)

Google generates more than 2 billion container deployments a week, all powered by its internal platform, Borg. Borg was the predecessor to Kubernetes, and the lessons learned from developing Borg over the years became the primary influence behind much of Kubernetes technology.

By deploying Kubernetes on a machine, a cluster will initially be created: it will be the case that wrap the overall infrastructure and components. The Kubernetes cluster is composed by two main kind of entities: Master and Nodes. The Master represents the control plane. It will be composed by some components that will be used, from the outside (the user), to interact with the cluster. These components represent the main contact point between the user and the cluster itself. On the other side, Nodes are the place where the business logic will run (the containerized user's apps). To summarize, the containerized applications will run inside each node and will be controlled through the master-node.

Both master and nodes are composed by many different components that need a little overview to understand as best as possible their role and functionalities.

Master Node is composed of elements which operate as the main management contact point for users, and, as shown in the Figure 2.1, it consists of the following components:

- **Etcd**: is a consistent and highly-available key-value map store used as K8S backing store for all the data used by the cluster. In other terms, it is a database that will be inside the cluster, used to reflect the state of the kubernetes cluster itself.

- **API Server**: is what exposes all the APIs developed by the Kubernetes comunity. These APIs represent the methods offered to the user to interact with the cluster itself. In simple terms, it is the management core of the entire cluster. It is a sort of bridge between the input commands and the various components that compose the cluster. It is the facade of the Kubernetes control plane.

- **Controller Manager**: it will manage the state of the cluster and regulate its life cycle. It is also responsible of performing routine tasks. For example, if we have defined a service with a certain number of replicas, the replication controller will try to understand if the replicas matches the number currently deployed on the cluster.

- **Scheduler Service**: is what manage the nodes workloads. To summarize, it is responsible of placing the workload on an acceptable node(s): it will try to determine which pods have to be placed inside each node, according to the scheduling queue.

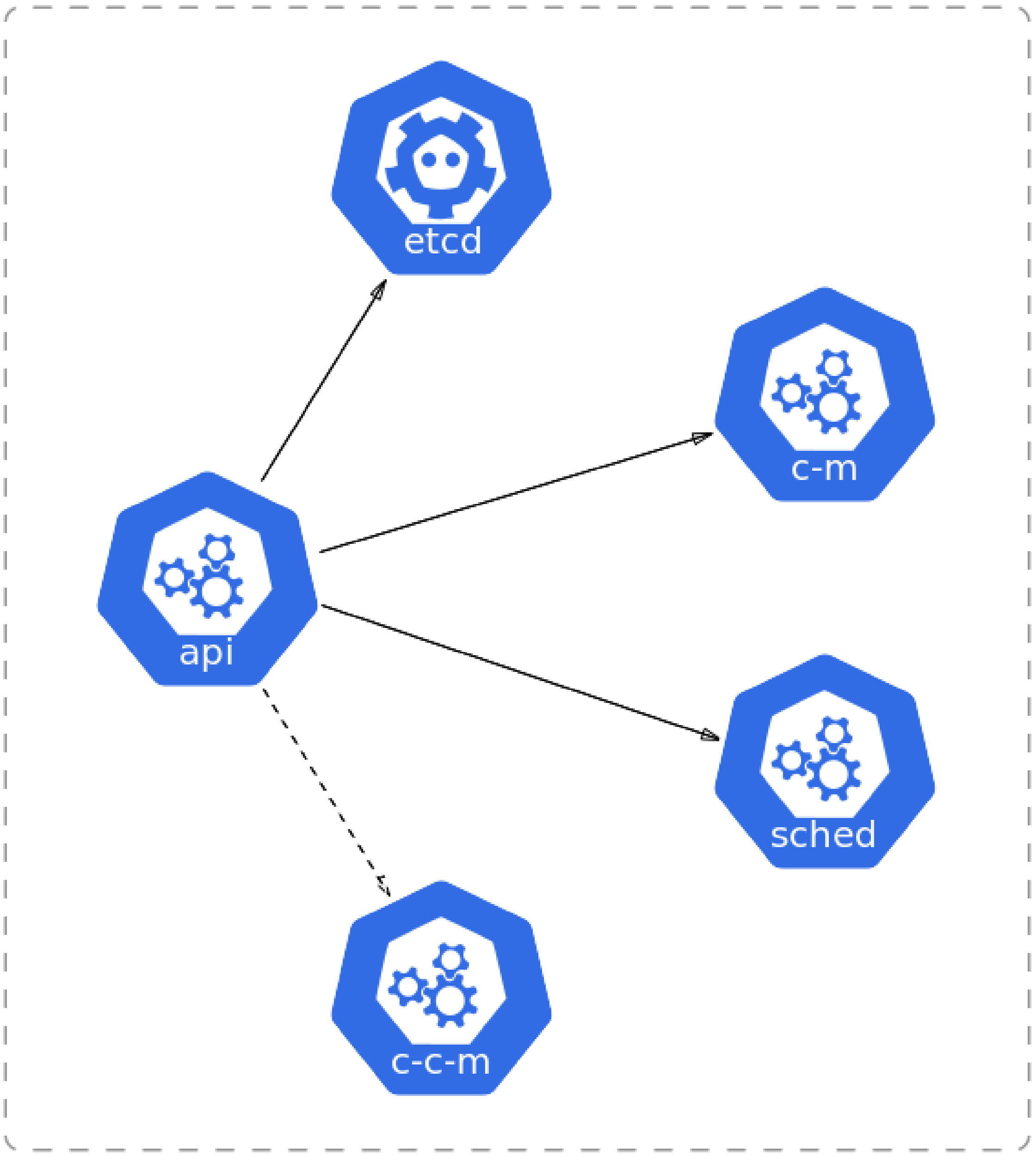

Figure 2.1: Master Plane

Node is the place in which the containerized applications will run. It is composed by the following elements[2]:

- **Docker**[3]: is the virtualization engine used to run all the applications.
- **Kubelet**: is the main contact point between the node and the control plane

services (the ones managed by the master-node).

- **Proxy**: it is used for reflecting the networking rules defined in the cluster and performing connection forwarding.

2.2 Cloud vs. On-Premise

There is a huge difference between cloud solutions and the on-premise ones. First we have to understand what cloud actually is and what benefits we obtain from this approach.

So, cloud is a big data center with lots of servers hosted in one place and interconnected with each other and managed by a cloud provider company. A software developer company can rent a portion of these resources, like CPU power, RAM, SSD, etc. for instance, in a form of VDS. Similarly, one can rent a Kubernetes Cluster, which has an autoscaling feature that is really crucial for applications with high load.

On the other hand, one can deploy one's application on-premises, handling the infrastructure issues on one's own, including the administrating, monitoring, updating the software, replacing the outdated components, and etc.

Both approaches have their own pro's and con's depending on concrete requirements for the application and infrastructure.

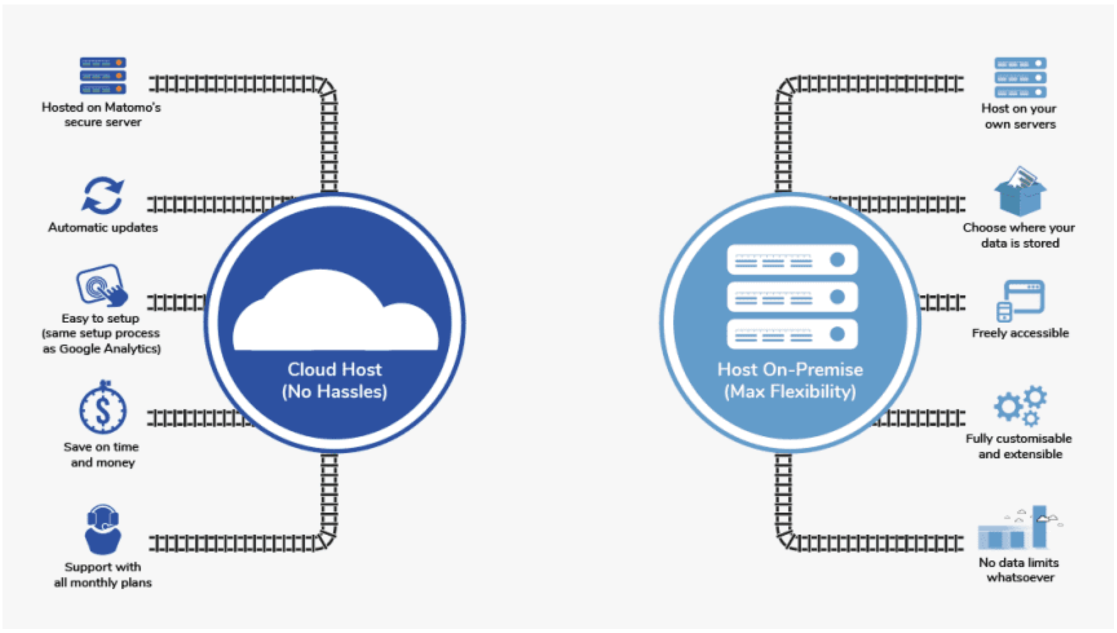
Hosted on Matomo's secure server
Automatic updates
Easy to setup (same setup process as Google Analytics)
Save on time and money
Support with all monthly plans
Cloud Host (No Hassles)
Host On-Premise (Max Flexibility)
Host on your own servers
Choose where your data is stored
Freely accessible
Fully customisable and extensible
No data limits whatsoever

Figure 2.2: Cloud vs. On-Premise

Cloud advantages:

- no need to buy a bare metal.
- administrating is delegated to a cloud provider
- most of required solutions come out-of-the-box
- high availability
- easy scalability

Cloud drawbacks:

- tight up to a 3rd party solution
- harder to customize
- data storage security issues due to some regulators requirements

On-Premise advantages:

- complete control over the whole infrastructure
- long-term cost efficiency
- in-house data storage

On-Premise drawbacks:

- maintenance cost
- implementation speed
- deployment process is handled in-house

2.3 Stateful vs. Stateless

Most of today's applications are stateless, because this architecture is easier to implement and maintain. You do not need to keep track of a state of each request, you just implement idempotent API methods and go to production without caring about sessions. Let us see what are the main differences between these two architectures.

Stateful Architecture

A stateful architecture or application is a framework that enables online users to store, record, and access previously completed tasks and information. It involves doing transactions while using earlier transactions as a guide. The previous transaction may have an impact on the present transaction in stateful applications.

Because of this, a stateful application processes its requests on the same server. Stateful transactions can be compared to a conversation where comments are made based on known facts. In circumstances where there is an incomplete transaction, you can pick up where you left off with stateful transactions.

A stateful application preserves the state of every session, regardless of its significance. Several current technologies today are built on stateful architecture. Both Telnet and the File Transfer Protocol (FTP) are excellent instances of stateful design. Email and online banking are two extremely important apps that leverage stateful design. It has some key advantages:

- stateful design is simple to understrand due to memory binding
- Because the stateful protocol saves data that aids subsequent transactions, it can give greater speed.
- Due to its great additional security layer, stateful architecture is highly well-liked in the banking and financial industry for use in online transactions.

There are a few drawbacks one should be aware of as well:

- The server design must incorporate memory for data storage. Stateful applications need a complex server since it exerts a heavy strain on the functionality of the entire program.
- Efficiency of the network memory has an impact on performance.
- This refers to ongoing management while the service is being provided.

Stateless Architecture

When using an Internet protocol, a stateless architecture or application does not keep or use the state of earlier transactions as a reference point. Each request that is communicated back and forth between parties may be understood and carried out without the requirement for prior requests. A client and server request and response are made using this protocol in the present state. Additionally, the current session's state is neither kept or carried over to the subsequent transaction.

Stateless apps use print servers and a Content Delivery Network to handle urgent demands (CDN). Sending an SMS is an example of stateless protocol in action. The Hypertext Transfer Protocol (HTTP), the Domain Name System (DNS), and other protocols are examples of stateless protocols. This architecture has some key advantages:

- It reduces the amount of resources, like as storage, that would otherwise be required to keep transactions active.
- Since no state is kept or required to be retained, stateless protocols can quickly recover from system failure.
- Stateless architecture is readily scaleable up or down, depending on the situation, without losing functionality.

There are a few drawbacks one should be aware of as well:

- Due to the volume of data sent out repeatedly, network performance may suffer.
- Since there is no information storage, stateless architecture is less competent to do specific tasks.

Important Comparisons Between Stateful and Stateless Architectures

We use both stateful and stateless designs on the internet every day. However, they exist in many architectures and are utilized in various applications. This is as a result of the two protocols' differences:

1 **Preserving data on servers**

  The most obvious distinction between stateless and stateful protocols, as well as the architecture on which they are based, is how both protocols manage data. Data storage is not a top priority for the stateless protocol. As a result, the servers that make up the network's architecture do not need to be designed to store a lot of data.

  Data is not transitory and does not need to be kept indefinitely on the servers when using stateless protocols. The client, which stores data as a cache, is primarily responsible for preserving information. Restarting the server also entails merely beginning a fresh process with no substantial data loss.

  Stateful applications, on the other hand, require a server with a lot of data storage space. Managing an application's whole life cycle that makes use of the stateful protocol may be somewhat difficult. The usage of the appropriate backing storage must also be ensured by the administrators. The stateful protocol mandates that servers preserve data from active transactions so that it may be referred to in subsequent transactions.

2 Implementation simplicity

Some protocols are simpler to implement than others when it comes to the internet and the realm of data transmission. The general classifications of stateful and stateless protocols fall under the same umbrella. To ascertain the nature of the request, stateless protocols require less logical reasoning, storage, and queries. Stateless applications are consequently simpler to design and frequently demand less computer processing power.

Because stateful applications demand more computer processing power and storage space than stateless ones, stateful protocols vary from stateless protocols in this way. Stateful protocols are more conceptually complex and difficult to implement than stateless ones.

3 Client-Server relationship

A two-way interface is often needed for computer programs to exchange data. For instance, a phone cannot browse the internet on its own unless it is linked to a server. The server then mediates this request after receiving requests from the client. This theory is applicable to websites, programs, the cloud, databases, etc.

The level of dependence between servers and client hardware, however, varies from protocol to protocol. There is less dependence between servers and clients in stateless protocols. Sending requests reduces the server's workload because they are self-contained.

Stateful protocols do, however, maintain a high degree of client-server interaction. Before users may create a connection, the server must react to requests issued by clients. Resubmitting the request is required if the client does not get a response from the server. This demonstrates how dependent clients and servers are on one another in stateful architecture.

4 Controlling system failures

The way the stateful and stateless protocols react to partial or total system failures is another distinction between them. When not managed appropriately, system failure caused by software or hardware components can have severe effects. The application's protocol still dictates how to respond to a system failure even with correct handling.

Stateless protocols don't require servers to store session data, and data transmitted in earlier transactions doesn't have much of an impact on ongoing sessions. As a result, a failed server may easily be restarted following a crash with minimum data loss because the system is not required to maintain any particular state.

This is in stark contrast to the stateful protocol, which keeps records of both current and past sessions in a certain state. In a stateful architecture, the server is required to keep track of all of its details and status information. Data loss occurs as a result of a server crash because lost data cannot be recovered even when the server is restarted.

5 Server complexity

In the past, servers were designed to handle the majority of processing needs for linked devices. These devices had hardware and software limitations, which allowed the extremely sophisticated servers to handle most of the processing and storage. The ultra-fast, high-capacity devices in use today have favorably contributed to making servers less sophisticated than they formerly were.

The same is true for stateless protocols, which rely less on their servers and, thus, don't need as sophisticated of servers to run. The server design is fairly straightforward because to the architecture of stateless protocols.

On the other hand, stateful protocol design keeps the practice of releasing the client device while transferring the majority of the burden to the server. As a result, stateful protocol servers need to be built with a major emphasis on complexity.

6 Scalability

When determining what kind of architecture to use for any purpose, scalability is a factor of growth that must be taken into account. Increased metric traffic may cause congestion and the need to increase the capacity of an application or website. This often entails adding more services to container orchestration for cloud-based servers or apps.

Scaling up or down is simple and may be carried out automatically for cloud-based apps utilizing an auto-scaler tool under the stateless service architecture. A front-end load balancer may be expanded to include back-end servers, and every server is capable of handling requests.

When it comes to scalability, a stateful protocol takes a different approach. In a stateful architecture, one must manually add more stateful servers and services to the current services in order to scale up. The same holds true when services are scaled back.

7 Transaction latency

In the modern world, speed is still one of the most important factors taken into account when considering any function or service. While this may be due to a number of factors, the protocol connecting the program also affects how quickly transactions happen. Some apps are just naturally quicker than others.

In stateless apps, no session data is stored on the server. It can also execute numerous sessions concurrently without requesting further information from a storage platform. This makes it possible for a stateful protocol to process incoming requests and transactions quickly.

However, stateful routing is configured to force the server to keep track of transactions in sessions for the duration of the session. Higher control over the information that is transferred across the network and being passed thanks to this type of transaction processing. The server's ability to process transactions per second is nonetheless limited. By compromising the speed of routine operations on the protocol, a higher level of control over transactions is achieved.

8 Multitasking

In terms of computer technology, multitasking refers to a system's capacity to concurrently handle numerous data inputs and create information. As a result of a server's capacity to handle several requests, multitasking occurs. A stateless protocol doesn't rely on a server in any way. Each request stands alone and is unrelated to previous transactions. Since there is no requirement to retrieve stored data, any resource that is available can handle the request.

In contrast, stateful applications work the other way around. The same stateful resource or server must be used for all transactions inside a session. The information that will be utilized for subsequent transactions is already on the server that was first used, therefore usage is limited to the duration of the session.

9 Response design

Both stateless and stateful protocols have differences in the way devices communicate and react to queries. For instance, in the stateless design, the client contacts the server with a request. The client does not check to see if the message has been received after it has been delivered. For instance, sending an SMS from a mobile phone doesn't need confirmation.

It does not anticipate a response once it has been sent. A stateful protocol, on the other hand, needs a relationship between requests and answers in order for transactions to be successful. A request is sent back to the server if it goes unanswered.

10 Similarities between use cases connected to firewalls

A firewall is a network security tool that controls and tracks traffic entering and leaving a network in accordance with the security protocol already established by the company. It serves as a partition between a company's private network and the wider public internet network.

Stateless protocols were first used to establish firewalls. They act to filter packets moving across stateless networks together with a typical access control list on layer 3 switches and routers. This was accomplished by looking at every packet to determine the source and destination IP addresses that were contained in the header. The firewall will let the packet through if it comes from the correct source.

This is carried out for each incoming packet in the stateless protocol. These days, stateful protocols are often used to build firewalls. They use an active connection table to keep track of the port information in addition to the source and destination IP addresses.

Stateful firewall services examine a packet's contents to make sure the right information is being transmitted. Stateful connections can identify data in a packet as coming from a source that has already been given permission to pass through the firewall because they retain data that is needed in later transactions.

A stateful firewall may filter data packets from unauthorized networks in addition to doing this. Stateful and stateless architecture both work in a similar manner to prevent malicious or unverified data packets from entering the network.

11 Similarities between use cases connected to databases

Many organizations and businesses of all sorts utilize database systems to store lengthy information on their clients, activities, etc. In a stateless application, no data or client-specific information is kept on the server, freeing it up for other tasks. They do, however, have a method of archiving data, which is accomplished with the use of database management systems.

A load balancer on a network directs traffic to a server as it enters the network. The user then contacts that server with his request. The stateless protocol creates an authentication token and keeps it in the database system together with the client's data. The token is returned to the front end so that the server may compare it to the data in the database during subsequent queries.

As the server does not keep information, this preserves the independent state of each request.

The stateful protocol also uses database management systems. Despite the server's capacity for data storage, the database continues to function as the principal repository for data. While the server retains the data transmitted over a session of numerous transactions, the database serves as the back end for stateful architecture.

2.4 Publish-Subscriber pattern

If referring to most cited definitions by Matthew O'Riordan in his article titled "Everything You Need To Know About Publish/Subscribe" [4] we can understand that:

"The Publish/Subscribe pattern, also known as pub/sub, is an architectural design pattern that provides a framework for exchanging messages between publishers and subscribers. This pattern involves the publisher and the subscriber relying on a message broker that relays messages from the publisher to the subscribers. The host publishes messages to a channel that subscribers can then sign up to."

In other words, this design approach allows an application to asynchronously broadcast events to many interested subscribers when senders and receivers are decoupled by a message broker.

The broker is the server-side engine that controls topics and resource availability; it is typically backed by a zookeeper service that works to keep resources operational. The subject is a simple tool for distributing messages and event alerts throughout the system. As we shall see later, there are several alternative policies that can be used to determine how the subscribers will consume from the topic, making it slightly different from a traditional message queue in that there may be numerous subscribers interested in the same message on the same topic.

The publisher is the element that creates messages on a certain subject, and the subscriber is the component that listens to a given subject to read messages from it.

The strongest and most important feature of this pattern is that Publisher and Subscriber (we could have many different publishers and subscribers) are completely unaware of one another's identities. This feature enables asynchronous broadcast of a collection of messages to various system components. The publisher and the customer will be completely independent of one another, as is clear to grasp.

There is a bunch of implementations of publish-subscriber architecture and in this particular work we will study one of the most popular framework called **Redis**.

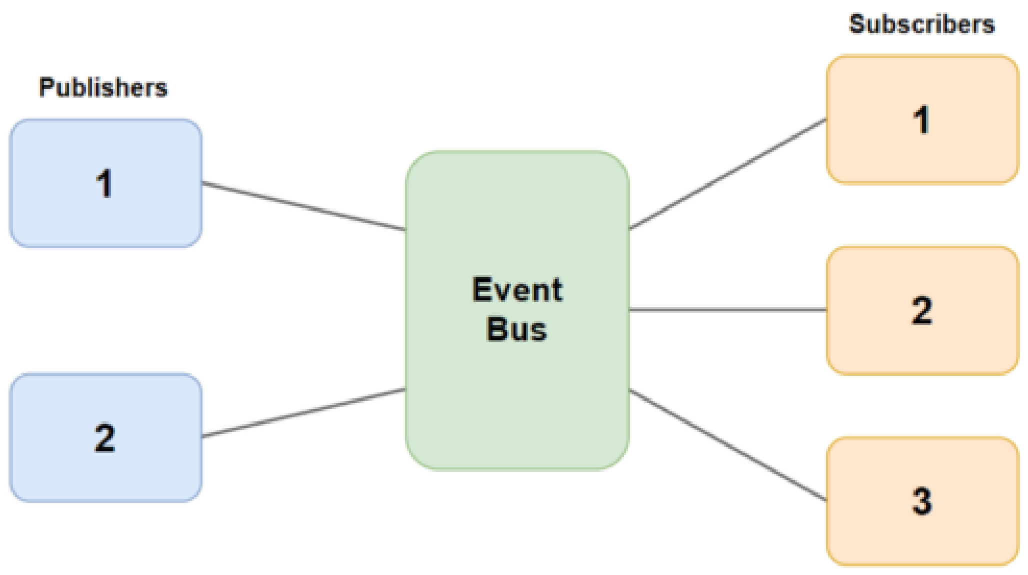

Figure 2.3: Publish-Subscriber architecture

Redis is the "Swiss Army knife" of in-memory databases, supporting a wide range of data types. It's frequently used for caching, but it may also be used for other purposes. It may also serve as a loosely linked distributed message broker.

Redis utilizes a data type called a "channel" that allows common pub/sub operations like publish and subscribe. Redis Pub/Sub is the oldest messaging pattern supported by Redis. Because publishers and subscribers are unaware of one another, it is regarded as being loosely connected. Subscribers can access one or more channels that have been subscribed to by publishers, who post messages to one or more channels.

The messages are sent to all currently connected subscribers on a channel, which may have zero or more subscribers. As a result, Redis Pub/Sub is adaptable and supports a variety of topologies, such as fan-in (many producers, single subscriber), fan-out (single producer, multiple subscribers), and 1-1. (one producer, one consumer).

This looks like a very standard pub/sub system so far, but it's critical to emphasize one feature: "connected" delivery semantics.

Connected Delivery Semantics

Connected delivery works similarly to radio. Radio stations transmit continuously on several frequencies (channels), but listeners can only pick up the broadcast when their receiver is plugged in, turned on, and tuned to a station.

Delivering in a connected way means:

1. Messages are only delivered to connected subscribers.

2. Each message is received by each subscriber that is connected.

3. There is no "memory" in the system once the message has been sent to all existing subscribers; it is immediately destroyed.

It follows that:

1. A subscriber who unsubscribes (disconnects) from a channel and then resubscribes to it will not get any of the messages it missed while offline and won't know whether it missed any messages. The message will simply be deleted and not sent to any subscribers if there are currently no subscribers to the channel.

2. The semantics of delivery are thus "at-most-once" per subscriber.

3. Considering that the message must be sent to all current subscribers before being deleted:

   (a) With additional subscriptions, this will take longer.

   (b) Contrary to radio broadcasts, which send information to every listener within range at the speed of light, this one does not.

"Disconnection" is intended, but it can also be caused by client or network issues, which could be unexpected and result in message loss.

According to the Redis Pub/Sub documentation and other sources, Redis employs push notifications to make sure messages are sent to all existing subscribers, which might have a negative impact on performance for big subscriber numbers.

Application examples of Redis Pub/Sub

1. Real-time, low-latency, urgent messages: since messages age quickly and have a limited shelf life, they are only relevant to subscribers temporarily.

2. Unreliable delivery/lossy messaging: If certain messages are simply ignored as a result of unreliable delivery (redundant messages of low significance as opposed to singularly vital "business" communications), it doesn't matter. Messages may be rejected as a result of network and subscriber failures, as well as failover from master to replicas.

3. A requirement on distribution that can only happen once per subscriber (subscribers are not capable of detecting duplicate messages and target systems are not idempotent.)

4. If subscribers only want to hear from a certain channel for a limited amount of time and have a transient, changing, or dynamic interest in the channel. For instance, mobile IoT devices could only be sporadically connected and only be interested in and able to respond to messages that are now being sent to them nearby.

Chapter 3

Infrastructure Setup

3.1 Requirements

The requirements for the infrastructure to be on-premises quite trickie. Let us discuss the main criteria that impact on the decision we make. So, we have the following impact domains[5]:

- **Security:** if we handle sensitive user data and there is a regulator mandate to not pass it to third parties, we are oblijed to not use cloud.
- **Economic:** If the cost of cloud rate is not well-suited in a financial model of the company, we are again looking towards the on-premises solutions
- **Maintenance:** If we do not have enough professionals like System Administrators and DevOps engineers, it is quite risky to launch this campaign and it is better leaning towards Cloud solutions.
- **Control:** If your application does not require sophisticated infrastructure solutions and standard, out-of-the-box cloud producrs are more than enough, you better stick with it.

From this perspective, we can assume that there is a bunch of variations when we have to get our on-premises infrastructure up and running, or at least to make a hybrid setup.

3.2 Kubernetes installation

This experiment was held on Google Cloud Platform, but without renting the kubernetes out-of-the-box. Instead, 6 virtual machines were dedicated for rolling out a Kubernetes Cluster on them. 3 VMs were for Control Plane, 3 replicas for redundancy, and similarly, 3 VMs for worker nodes with the same replication strategy. The overall construction suggested by Kelsey Hightower[6] looks like this:

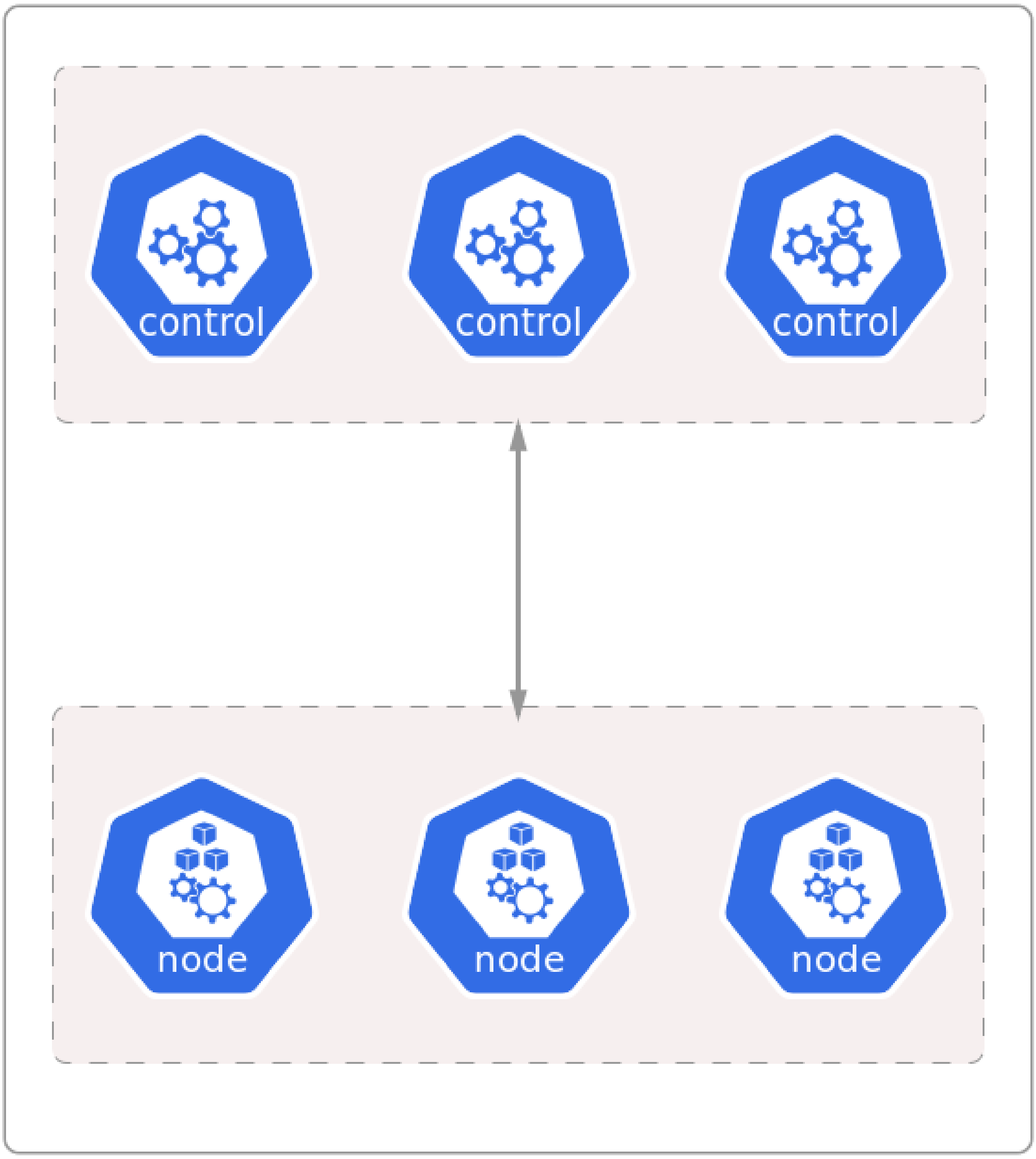

Figure 3.1: Custom Infrastructure

For the Kubernetes control plane and the worker nodes, where containers are finally operated, Kubernetes needs a group of virtual machines[7]. In this experiment, we will set up the computing resources needed to run a Kubernetes cluster securely and with high availability across a single compute zone.

Networking

The networking paradigm used by Kubernetes presupposes a flat network where nodes and containers may talk to one another. Network policies can restrict how groups of containers are permitted to communicate with one another and external network endpoints in situations when this is not desirable.

We have to create a VPC network to host the Kubernetes Cluster

```
 $ gcloud compute networks create kubernetes-on-premise --subnet-
  mode custom
```

A subnet must be configured with an IP address range big enough to provide each node in the Kubernetes cluster a private IP address.

We create the kubernetes subnet in the kubernetes-on-premise VPC network:

```
 $ gcloud compute networks subnets create kubernetes \
--network kubernetes-on-premise \
--range 10.240.0.0/24
```

Firewall

Make a firewall rule that permits all internal protocols of communication:

```
 $ gcloud compute firewall-rules create kubernetes-on-premise-allow-
   internal \
--allow tcp,udp,icmp \
--network kubernetes-on-premise \
--source-ranges 10.240.0.0/24,10.200.0.0/16
```

Make a firewall rule that permits HTTPS, ICMP, and SSH from outside sources:

```
 $ gcloud compute firewall-rules create kubernetes-on-premise-allow-
   external \
--allow tcp:22,tcp:6443,icmp \
--network kubernetes-on-premise \
--source-ranges 0.0.0.0/0
```

Public IP Address

Create and bind a static IP address to the external load balancer fronting the Kubernetes API Servers so that it may be connected to it:

```
$ gcloud compute addresses create kubernetes-on-premise \
--region $(gcloud config get-value compute/region)
```

Create Instances

With its strong support for the containerd container runtime, Ubuntu Server 20.04 will be used to supply the compute instances in this experiment. To make the Kubernetes bootstrapping process simpler, a fixed private IP address will be assigned to each compute instance.

Now we create three compute instances for control plane:

```
$ for i in 0 1 2; do
gcloud compute instances create controller-${i} \
  --async \
  --boot-disk-size 200GB \
  --can-ip-forward \
  --image-family ubuntu-2004-lts \
  --image-project ubuntu-os-cloud \
  --machine-type e2-standard-2 \
  --private-network-ip 10.240.0.1${i} \
  --scopes compute-rw,storage-ro,service-management,service-control
  ,logging-write,monitoring \
  --subnet kubernetes \
  --tags kubernetes-on-premise,controller
done
```

And three compute instances for the worker nodes:

```
$ for i in 0 1 2; do
gcloud compute instances create worker-${i} \
  --async \
  --boot-disk-size 200GB \
  --can-ip-forward \
  --image-family ubuntu-2004-lts \
    --image-project ubuntu-os-cloud \
    --machine-type e2-standard-2 \
    --metadata pod-cidr=10.200.${i}.0/24 \
    --private-network-ip 10.240.0.2${i} \
    --scopes compute-rw,storage-ro,service-management,service-control
    ,logging-write,monitoring \
    --subnet kubernetes \
    --tags kubernetes-on-premise,worker
done
```

Next, we will generate TLS certificates for kubernetes components like kube-proxy, kube-controller-manager, kube-scheduler, and others. After that configuration files for authentication are generated. And right before installing etcd cluster, we generate the Data Encryption Config and Key.

etcd Cluster

The commands are executed from inside of the controller instances so:

```
  $ gcloud compute ssh controller-0
  $ wget -q --show-progress --https-only --timestamping \
  "https://github.com/etcd-io/etcd/releases/download/v3.4.15/etcd-v3
  .4.15-linux-amd64.tar.gz"
  $ tar -xvf etcd-v3.4.15-linux-amd64.tar.gz
  $ sudo mv etcd-v3.4.15-linux-amd64/etcd* /usr/local/bin/
```

Now we configure the etcd Server:

```
  $ sudo mkdir -p /etc/etcd /var/lib/etcd
  $ sudo chmod 700 /var/lib/etcd
  $ sudo cp ca.pem kubernetes-key.pem kubernetes.pem /etc/etcd/
```

Serving client requests and interacting with etcd cluster peers will both be done using the instance's internal IP address. WE have to get the current compute instance's internal IP address:

```
  $ INTERNAL_IP=$(curl -s -H "Metadata-Flavor: Google" \
  http://metadata.google.internal/computeMetadata/v1/instance/network
   -interfaces/0/ip)
```

Within an etcd cluster, each etcd member has to have a distinctive name. Set the hostname of the current compute instance as the etcd name:

```
  $ ETCD_NAME=$(hostname -s)
```

Now we create the `etcd.service` systemd unit file:

```
  $ cat <<EOF | sudo tee /etc/systemd/system/etcd.service
[Unit]
Description=etcd
Documentation=https://github.com/coreos

[Service]
Type=notify
ExecStart=/usr/local/bin/etcd \\
  --name ${ETCD_NAME} \\
  --cert-file=/etc/etcd/kubernetes.pem \\
  --key-file=/etc/etcd/kubernetes-key.pem \\
  --peer-cert-file=/etc/etcd/kubernetes.pem \\
  --peer-key-file=/etc/etcd/kubernetes-key.pem \\
  --trusted-ca-file=/etc/etcd/ca.pem \\
  --peer-trusted-ca-file=/etc/etcd/ca.pem \\
  --peer-client-cert-auth \\
  --client-cert-auth \\
  --initial-advertise-peer-urls https://${INTERNAL_IP}:2380 \\
  --listen-peer-urls https://${INTERNAL_IP}:2380 \\
  --listen-client-urls https://${INTERNAL_IP}:2379,https
    ://127.0.0.1:2379 \\
  --advertise-client-urls https://${INTERNAL_IP}:2379 \\
  --initial-cluster-token etcd-cluster-0 \\
  --initial-cluster controller-0=https://10.240.0.10:2380,controller
    -1=https://10.240.0.11:2380,controller-2=https://10.240.0.12:2380
    \\
  --initial-cluster-state new \\
  --data-dir=/var/lib/etcd
Restart=on-failure
RestartSec=5

[Install]
```

```
WantedBy=multi-user.target
EOF
```

And finally start the etcd Server:

```
  $ sudo systemctl daemon-reload
  $ sudo systemctl enable etcd
  $ sudo systemctl start etcd
```

Kubernetes Control Plane

The Kubernetes control plane will be set up for high availability and bootstrapped across three compute instances. Additionally, a third-party load balancer that makes the Kubernetes API Servers accessible to outside clients will be built. Each node will have the Kubernetes API Server, Scheduler, and Controller Manager installed on it.

As in the previous part, all commands are executed from the inside of the controller instances:

```
  $ gcloud compute ssh controller-0
  $ sudo mkdir -p /etc/kubernetes/config
```

Install the Kubernetes Controller Binaries

```
  $ wget -q --show-progress --https-only --timestamping \
  "https://storage.googleapis.com/kubernetes-release/release/v1.21.0/
    bin/linux/amd64/kube-apiserver" \
  "https://storage.googleapis.com/kubernetes-release/release/v1.21.0/
    bin/linux/amd64/kube-controller-manager" \
  "https://storage.googleapis.com/kubernetes-release/release/v1.21.0/
    bin/linux/amd64/kube-scheduler" \
  "https://storage.googleapis.com/kubernetes-release/release/v1.21.0/
    bin/linux/amd64/kubectl"
```

Followed by:

```
$ chmod +x kube-apiserver kube-controller-manager kube-scheduler
  kubectl
$ sudo mv kube-apiserver kube-controller-manager kube-scheduler
  kubectl /usr/local/bin/
```

Now we have configure the Kubernetes API Server:

```
$ sudo mkdir -p /var/lib/kubernetes/
$ sudo mv ca.pem ca-key.pem kubernetes-key.pem kubernetes.pem \
  service-account-key.pem service-account.pem \
  encryption-config.yaml /var/lib/kubernetes/
```

Create the `kube-apiserver.service` file:

```
$ INTERNAL_IP=$(curl -s -H "Metadata-Flavor: Google" \
http://metadata.google.internal/computeMetadata/v1/instance/network
  -interfaces/0/ip)
$ REGION=$(curl -s -H "Metadata-Flavor: Google" \
http://metadata.google.internal/computeMetadata/v1/project/attributes
  /google-compute-default-region)
$ KUBERNETES_PUBLIC_ADDRESS=$(gcloud compute addresses describe
  kubernetes-on-premise \
--region $REGION \
--format 'value(address)')
```

Followed by:

```
$ cat <<EOF | sudo tee /etc/systemd/system/kube-apiserver.service
[Unit]
Description=Kubernetes API Server
Documentation=https://github.com/kubernetes/kubernetes

[Service]
ExecStart=/usr/local/bin/kube-apiserver \\
  --advertise-address=${INTERNAL_IP} \\
  --allow-privileged=true \\
  --apiserver-count=3 \\
  --audit-log-maxage=30 \\
  --audit-log-maxbackup=3 \\
  --audit-log-maxsize=100 \\
  --audit-log-path=/var/log/audit.log \\
  --authorization-mode=Node,RBAC \\
  --bind-address=0.0.0.0 \\
  --client-ca-file=/var/lib/kubernetes/ca.pem \\
  --enable-admission-plugins=NamespaceLifecycle,NodeRestriction,LimitRanger,ServiceAccount,DefaultStorageClass,ResourceQuota \\
  --etcd-cafile=/var/lib/kubernetes/ca.pem \\
  --etcd-certfile=/var/lib/kubernetes/kubernetes.pem \\
  --etcd-keyfile=/var/lib/kubernetes/kubernetes-key.pem \\
  --etcd-servers=https://10.240.0.10:2379,https://10.240.0.11:2379,https://10.240.0.12:2379 \\
  --event-ttl=1h \\
  --encryption-provider-config=/var/lib/kubernetes/encryption-config.yaml \\
  --kubelet-certificate-authority=/var/lib/kubernetes/ca.pem \\
  --kubelet-client-certificate=/var/lib/kubernetes/kubernetes.pem \\
  --kubelet-client-key=/var/lib/kubernetes/kubernetes-key.pem \\
  --runtime-config='api/all=true' \\
  --service-account-key-file=/var/lib/kubernetes/service-account.pem \\
  --service-account-signing-key-file=/var/lib/kubernetes/service-account-key.pem \\
  --service-account-issuer=https://${KUBERNETES_PUBLIC_ADDRESS}:6443 \\
  --service-cluster-ip-range=10.32.0.0/24 \\
  --service-node-port-range=30000-32767 \\
  --tls-cert-file=/var/lib/kubernetes/kubernetes.pem \\
  --tls-private-key-file=/var/lib/kubernetes/kubernetes-key.pem \\
  --v=2
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
EOF
```

Configuring the API Server

```
 $ sudo mkdir -p /var/lib/kubernetes/
 $ sudo mv ca.pem ca-key.pem kubernetes-key.pem kubernetes.pem \
   service-account-key.pem service-account.pem \
   encryption-config.yaml /var/lib/kubernetes/
```

Creating the `kube-apiserver.service` file:

```
  $ cat <<EOF | sudo tee /etc/systemd/system/kube-apiserver.service
[Unit]
Description=Kubernetes API Server
Documentation=https://github.com/kubernetes/kubernetes

[Service]
ExecStart=/usr/local/bin/kube-apiserver \\
  --advertise-address=${INTERNAL_IP} \\
  --allow-privileged=true \\
  --apiserver-count=3 \\
  --audit-log-maxage=30 \\
  --audit-log-maxbackup=3 \\
  --audit-log-maxsize=100 \\
  --audit-log-path=/var/log/audit.log \\
  --authorization-mode=Node,RBAC \\
  --bind-address=0.0.0.0 \\
  --client-ca-file=/var/lib/kubernetes/ca.pem \\
  --enable-admission-plugins=NamespaceLifecycle,NodeRestriction,LimitRanger,ServiceAccount,DefaultStorageClass,ResourceQuota \\
  --etcd-cafile=/var/lib/kubernetes/ca.pem \\
  --etcd-certfile=/var/lib/kubernetes/kubernetes.pem \\
  --etcd-keyfile=/var/lib/kubernetes/kubernetes-key.pem \\
  --etcd-servers=https://10.240.0.10:2379,https://10.240.0.11:2379,https://10.240.0.12:2379 \\
  --event-ttl=1h \\
  --encryption-provider-config=/var/lib/kubernetes/encryption-config.yaml \\
  --kubelet-certificate-authority=/var/lib/kubernetes/ca.pem \\
  --kubelet-client-certificate=/var/lib/kubernetes/kubernetes.pem \\
  --kubelet-client-key=/var/lib/kubernetes/kubernetes-key.pem \\
  --runtime-config='api/all=true' \\
  --service-account-key-file=/var/lib/kubernetes/service-account.pem \\
  --service-account-signing-key-file=/var/lib/kubernetes/service-account-key.pem \\
  --service-account-issuer=https://${KUBERNETES_PUBLIC_ADDRESS}:6443 \\
  --service-cluster-ip-range=10.32.0.0/24 \\
  --service-node-port-range=30000-32767 \\
  --tls-cert-file=/var/lib/kubernetes/kubernetes.pem \\
  --tls-private-key-file=/var/lib/kubernetes/kubernetes-key.pem \\
  --v=2
Restart=on-failure
RestartSec=5
```

```

[Install]
WantedBy=multi-user.target
EOF
```

Configuring the Controller Manager

```
  $ sudo mv kube-controller-manager.kubeconfig /var/lib/kubernetes/
```

Creating the `kube-controller-manager.service` file:

```
  $ cat <<EOF | sudo tee /etc/systemd/system/kube-controller-manager.
    service
[Unit]
Description=Kubernetes Controller Manager
Documentation=https://github.com/kubernetes/kubernetes

[Service]
ExecStart=/usr/local/bin/kube-controller-manager \\
  --bind-address=0.0.0.0 \\
  --cluster-cidr=10.200.0.0/16 \\
  --cluster-name=kubernetes \\
  --cluster-signing-cert-file=/var/lib/kubernetes/ca.pem \\
  --cluster-signing-key-file=/var/lib/kubernetes/ca-key.pem \\
  --kubeconfig=/var/lib/kubernetes/kube-controller-manager.kubeconfig
     \\
  --leader-elect=true \\
  --root-ca-file=/var/lib/kubernetes/ca.pem \\
  --service-account-private-key-file=/var/lib/kubernetes/service-
    account-key.pem \\
  --service-cluster-ip-range=10.32.0.0/24 \\
  --use-service-account-credentials=true \\
  --v=2
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
EOF
```

Configuring the Scheduler

```
  $ sudo mv kube-scheduler.kubeconfig /var/lib/kubernetes/
```

Creating the `kube-scheduler.yaml` file:

```
  $ cat <<EOF | sudo tee /etc/kubernetes/config/kube-scheduler.yaml
apiVersion: kubescheduler.config.k8s.io/v1beta1
kind: KubeSchedulerConfiguration
clientConnection:
  kubeconfig: "/var/lib/kubernetes/kube-scheduler.kubeconfig"
leaderElection:
  leaderElect: true
EOF
```

Creating the `kube-scheduler.service` file:

```
  $ cat <<EOF | sudo tee /etc/systemd/system/kube-scheduler.service
[Unit]
Description=Kubernetes Scheduler
Documentation=https://github.com/kubernetes/kubernetes

[Service]
ExecStart=/usr/local/bin/kube-scheduler \\
  --config=/etc/kubernetes/config/kube-scheduler.yaml \\
  --v=2
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
EOF
```

And finally we can start the Controller Services:

```
  $ sudo systemctl daemon-reload
  $ sudo systemctl enable kube-apiserver kube-controller-
    manager kube-scheduler
```

```
$ sudo systemctl start kube-apiserver kube-controller-manager
  kube-scheduler
```

Health Check Setup

Each of the three API servers will be able to terminate TLS connections and verify client certificates thanks to the usage of a Google Network Load Balancer, which will also distribute traffic across the servers. Because the network load balancer only supports HTTP health checks, the API server's HTTPS endpoint cannot be used. The nginx webserver can be used as a workaround to proxy HTTP health checks. Installed and set up in this part, nginx will receive HTTP health checks on port 80 and act as a proxy for connections to the API server at `https://127.0.0.1:6443/healthz`.

```
$ sudo apt-get update
$ sudo apt-get install -y nginx
```

```
  $ cat > kubernetes.default.svc.cluster.local <<EOF
server {
  listen      80;
  server_name kubernetes.default.svc.cluster.local;

  location /healthz {
     proxy_pass                    https://127.0.0.1:6443/healthz;
     proxy_ssl_trusted_certificate /var/lib/kubernetes/ca.pem;
  }
}
EOF
```

```
$ sudo mv kubernetes.default.svc.cluster.local \
  /etc/nginx/sites-available/kubernetes.default.svc.cluster.local
$ sudo ln -s /etc/nginx/sites-available/kubernetes.default.svc.
  cluster.local /etc/nginx/sites-enabled/
```

```
$ sudo systemctl restart nginx
$ sudo systemctl enable nginx
```

External Load Balancer

```
  $ KUBERNETES_PUBLIC_ADDRESS=$(gcloud compute addresses describe
   kubernetes-on-premise \
    --region $(gcloud config get-value compute/region) \
    --format 'value(address)')

  $ gcloud compute http-health-checks create kubernetes \
    --description "Kubernetes Health Check" \
    --host "kubernetes.default.svc.cluster.local" \
    --request-path "/healthz"

  $ gcloud compute firewall-rules create kubernetes-on-premise-allow-
   health-check \
    --network kubernetes-the-hard-way \
    --source-ranges 209.85.152.0/22,209.85.204.0/22,35.191.0.0/16 \
    --allow tcp

  $ gcloud compute target-pools create kubernetes-target-pool \
    --http-health-check kubernetes

  $ gcloud compute target-pools add-instances kubernetes-target-pool
   \
   --instances controller-0,controller-1,controller-2

  $ gcloud compute forwarding-rules create kubernetes-forwarding-rule
    \
    --address ${KUBERNETES_PUBLIC_ADDRESS} \
    --ports 6443 \
    --region $(gcloud config get-value compute/region) \
    --target-pool kubernetes-target-pool
}
```

Worker Nodes

Each node will have runc, container networking plugins, containerd, kubelet, and kube-proxy installed on it.

First, we install Operating System dependencies:

```
  $ gcloud compute ssh worker-0
  $ sudo apt-get update
  $ sudo apt-get -y install socat conntrack ipset
```

If swap is enabled, the kubelet will by default fail to start. Swap should be turned off to make sure Kubernetes can deliver adequate resource allocation and service quality.

```
$ sudo swapoff -a
```

Install the Woker Binaries

```
$ wget -q --show-progress --https-only --timestamping \
https://github.com/kubernetes-sigs/cri-tools/releases/download/v1
  .21.0/crictl-v1.21.0-linux-amd64.tar.gz \
https://github.com/opencontainers/runc/releases/download/v1.0.0-
  rc93/runc.amd64 \
https://github.com/containernetworking/plugins/releases/download/v0
  .9.1/cni-plugins-linux-amd64-v0.9.1.tgz \
https://github.com/containerd/containerd/releases/download/v1.4.4/
  containerd-1.4.4-linux-amd64.tar.gz \
https://storage.googleapis.com/kubernetes-release/release/v1.21.0/
  bin/linux/amd64/kubectl \
https://storage.googleapis.com/kubernetes-release/release/v1.21.0/
  bin/linux/amd64/kube-proxy \
https://storage.googleapis.com/kubernetes-release/release/v1.21.0/
  bin/linux/amd64/kubelet
```

```
$ sudo mkdir -p \
/etc/cni/net.d \
/opt/cni/bin \
/var/lib/kubelet \
/var/lib/kube-proxy \
/var/lib/kubernetes \
/var/run/kubernetes
```

```
$ mkdir containerd
$ tar -xvf crictl-v1.21.0-linux-amd64.tar.gz
$ tar -xvf containerd-1.4.4-linux-amd64.tar.gz -C containerd
$ sudo tar -xvf cni-plugins-linux-amd64-v0.9.1.tgz -C /opt/cni/bin/
$ sudo mv runc.amd64 runc
$ chmod +x crictl kubectl kube-proxy kubelet runc
$ sudo mv crictl kubectl kube-proxy kubelet runc /usr/local/bin/
$ sudo mv containerd/bin/* /bin/
```

CNI Networking[8]

```
$ POD_CIDR=$(curl -s -H "Metadata-Flavor: Google" \  http://
  metadata.google.internal/computeMetadata/v1/instance/attributes/
  pod-cidr)
```

Creating the `bridge` file:

```
$ cat <<EOF | sudo tee /etc/cni/net.d/10-bridge.conf
{
    "cniVersion": "0.4.0",
    "name": "bridge",
    "type": "bridge",
    "bridge": "cnio0",
    "isGateway": true,
    "ipMasq": true,
    "ipam": {
        "type": "host-local",
        "ranges": [
          [{"subnet": "${POD_CIDR}"}]
        ],
        "routes": [{"dst": "0.0.0.0/0"}]
    }
}
EOF
```

Creating the `loopback` file:

```
$ cat <<EOF | sudo tee /etc/cni/net.d/99-loopback.conf
{
    "cniVersion": "0.4.0",
    "name": "lo",
    "type": "loopback"
}
EOF
```

Creating the `containerd` file:

```
  $ sudo mkdir -p /etc/containerd/
  $ cat << EOF | sudo tee /etc/containerd/config.toml
[plugins]
  [plugins.cri.containerd]
    snapshotter = "overlayfs"
    [plugins.cri.containerd.default_runtime]
      runtime_type = "io.containerd.runtime.v1.linux"
      runtime_engine = "/usr/local/bin/runc"
      runtime_root = ""
EOF
```

Creating the `containerd.service` file:

```
  $ cat <<EOF | sudo tee /etc/systemd/system/containerd.service
[Unit]
Description=containerd container runtime
Documentation=https://containerd.io
After=network.target

[Service]
ExecStartPre=/sbin/modprobe overlay
ExecStart=/bin/containerd
Restart=always
RestartSec=5
Delegate=yes
KillMode=process
OOMScoreAdjust=-999
LimitNOFILE=1048576
LimitNPROC=infinity
LimitCORE=infinity

[Install]
WantedBy=multi-user.target
EOF
```

Kubelet Configuration:

```
  $ sudo mv ${HOSTNAME}-key.pem ${HOSTNAME}.pem /var/lib/kubelet/
  $ sudo mv ${HOSTNAME}.kubeconfig /var/lib/kubelet/kubeconfig
  $ sudo mv ca.pem /var/lib/kubernetes/
```

```
  $ cat <<EOF | sudo tee /var/lib/kubelet/kubelet-config.yaml
kind: KubeletConfiguration
apiVersion: kubelet.config.k8s.io/v1beta1
authentication:
  anonymous:
    enabled: false
  webhook:
    enabled: true
  x509:
    clientCAFile: "/var/lib/kubernetes/ca.pem"
authorization:
  mode: Webhook
clusterDomain: "cluster.local"
clusterDNS:
  - "10.32.0.10"
podCIDR: "${POD_CIDR}"
resolvConf: "/run/systemd/resolve/resolv.conf"
runtimeRequestTimeout: "15m"
tlsCertFile: "/var/lib/kubelet/${HOSTNAME}.pem"
tlsPrivateKeyFile: "/var/lib/kubelet/${HOSTNAME}-key.pem"
EOF
```

Creating the `kubelet.service` file:

```
  $ cat <<EOF | sudo tee /etc/systemd/system/kubelet.service
[Unit]
Description=Kubernetes Kubelet
Documentation=https://github.com/kubernetes/kubernetes
After=containerd.service
Requires=containerd.service

[Service]
ExecStart=/usr/local/bin/kubelet \\
  --config=/var/lib/kubelet/kubelet-config.yaml \\
  --container-runtime=remote \\
  --container-runtime-endpoint=unix:///var/run/containerd/containerd.
    sock \\
  --image-pull-progress-deadline=2m \\
  --kubeconfig=/var/lib/kubelet/kubeconfig \\
  --network-plugin=cni \\
  --register-node=true \\
  --v=2
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
EOF
```

Kubernetes Proxy Configuration:
Creating the `kube-proxy-config.yaml` file:

```
  $ sudo mv kube-proxy.kubeconfig /var/lib/kube-proxy/kubeconfig
  $ cat <<EOF | sudo tee /var/lib/kube-proxy/kube-proxy-config.yaml
kind: KubeProxyConfiguration
apiVersion: kubeproxy.config.k8s.io/v1alpha1
clientConnection:
  kubeconfig: "/var/lib/kube-proxy/kubeconfig"
mode: "iptables"
clusterCIDR: "10.200.0.0/16"
EOF
```

Creating the `kube-proxy.service` file:

```
  $ cat <<EOF | sudo tee /etc/systemd/system/kube-proxy.service
[Unit]
Description=Kubernetes Kube Proxy
Documentation=https://github.com/kubernetes/kubernetes

[Service]
ExecStart=/usr/local/bin/kube-proxy \\
  --config=/var/lib/kube-proxy/kube-proxy-config.yaml
Restart=on-failure
RestartSec=5

[Install]
```

```
WantedBy=multi-user.target
EOF
```

Start the Workers!

```
  $ sudo systemctl daemon-reload
  $ sudo systemctl enable containerd kubelet kube-proxy
  $ sudo systemctl start containerd kubelet kube-proxy
```

And the Last configuration, but not the least: The Admin Kubernetes Configuration

A Kubernetes API Server is needed to connect to each kubeconfig. The IP address assigned to the external load balancer fronting the Kubernetes API Servers will be utilized to enable high availability.

```
  $ KUBERNETES_PUBLIC_ADDRESS=$(gcloud compute addresses describe
    kubernetes-on-premise \
    --region $(gcloud config get-value compute/region) \
    --format 'value(address)')

  $ kubectl config set-cluster kubernetes-on-premise \
    --certificate-authority=ca.pem \
    --embed-certs=true \
    --server=https://${KUBERNETES_PUBLIC_ADDRESS}:6443

  $ kubectl config set-credentials admin \
    --client-certificate=admin.pem \
    --client-key=admin-key.pem

  $ kubectl config set-context kubernetes-on-premise \
    --cluster=kubernetes-the-hard-way \
    --user=admin

  kubectl config use-context kubernetes-on-premise
```

Now, if we run

```
  $ kubectl get nodes
```

we should be seeing

```
NAME        STATUS   ROLES    AGE     VERSION
worker-0    Ready    <none>   2m35s   v1.21.0
worker-1    Ready    <none>   2m35s   v1.21.0
worker-2    Ready    <none>   2m35s   v1.21.0
```

Chapter 4

Stateful Application on Kubernetes

4.1 StatefulSet

"StatefulSet is the workload API object used to manage stateful applications." is said in Kubernetes manuals[9].

StatefulSet is a twin brother of a Deployment but with its own uniqueness - it's created for deploying and handling stateful services and applications. It assigns a fixed unique identifier to a pod and manages pods in a strict ordered manner, i.e. it starts n-th pod assigning id: (n-1) and not the other one. Also when deleting it deletes only the pod with has the highest identifier value.

StatefulSet also has some limitations on usaage:

- Storage for a specific Pod must either be pre-provided by an administrator or provisioned by a persistent volume provisioner depending on the required storage type.

- The volumes linked to a StatefulSet won't be deleted if it is deleted or scaled down. Data security is ensured in this way, which is typically more beneficial than automatically deleting all associated StatefulSet resources.

- The network identification of the Pods must currently be managed by a Headless Service for StatefulSets. We are in charge of developing this Service.

- When a StatefulSet is removed, there are no promises made regarding the termination of any pods. It is possible to scale the StatefulSet down to 0 before deletion in order to provide an orderly and elegant end of the pods in the StatefulSet.

4.2 Sample application using StatefulSet

As it was declared about the limitations of StatefulSets, to create a working and responding component we have to create a headless Service component and devote 10 Gb of persistant memory to the StatefulSet[10]:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mongo
  labels:
    app: mongo
spec:
  ports:
  - port: 27017
    targetPort: 27017
    name: db
  clusterIP: None
  selector:
    app: mongo
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: db
spec:
  serviceName: "mongo"
  replicas: 2
  selector:
    matchLabels:
      app: mongo
  template:
    metadata:
      labels:
        app: mongo
    spec:
      containers:
      - name: mongo
        image: cvallance/mongo-k8s-sidecar
        ports:
        - containerPort: 27017
          name: db
        volumeMounts:
          - name: mongo-persistent-storage
            mountPath: /data/db
  volumeClaimTemplates:
    - metadata:
        name: mongo-persistent-storage
      spec:
        storageClassName: "fast"
        accessModes: ["ReadWriteOnce"]
        resources:
          requests:
            storage: 10Gi
```

4.3 BigBlueButton

BigBlueButton is an open source conferencing and learning platform which is has a software oriented architecture[11] and it needs refactoring to make it fit to Kubernetes orchestrating.

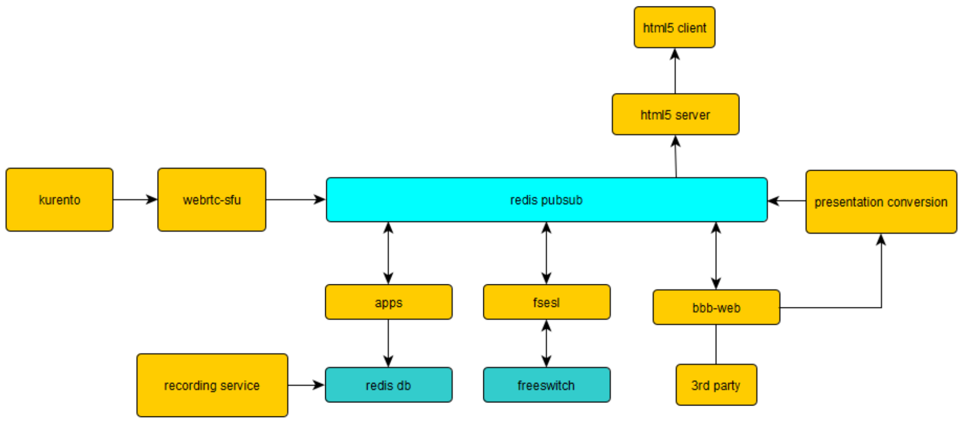

Figure 4.1: BigBlueButton architecture

The problem is that it is mostly stateful. For instance every time a request comes from the client, the state of that request is stored in MongoDB. There is also another stateful component which is already familiar to us - Redis is used as a Pub/Sub broker. And it fits here just fine with its "Connected Delivery Semantics".

Another issue is that `html5` service is a monolithic client-server component which leads to a hard to maintain problem, and normally should be decoupled to two separate front-end and back-end services.

From the network perspective, more detailed information is given in a figure below.

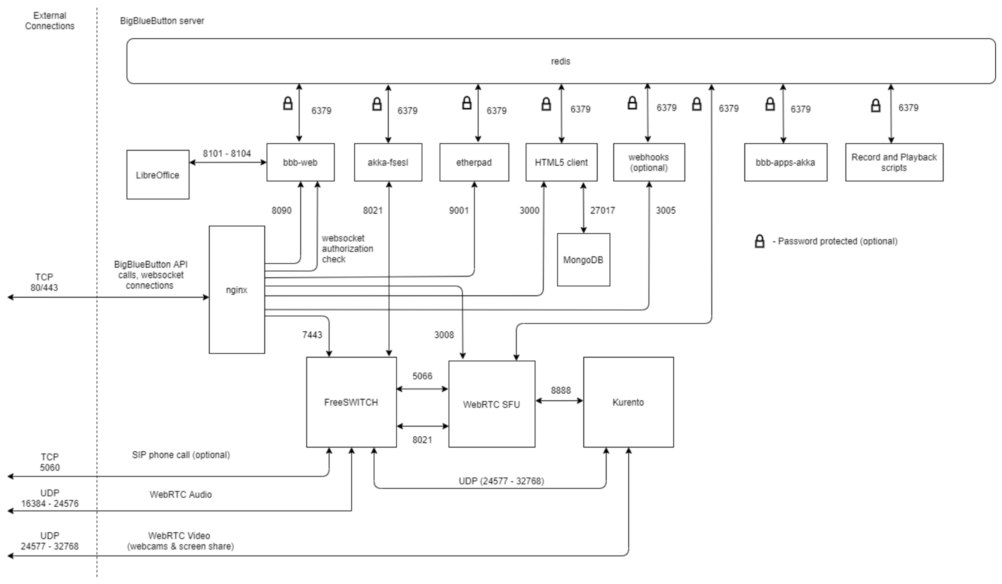

Figure 4.2: BigBlueButton service connections

4.4 BigBlueButton on Kubernetes

From the obtained information we have factor out the stateful components of the application and build a new Kubernetes based architecture:

- **Redis** is deployed as a StatefulSets
- **MongoDB** is deployed as a StatefulSets
- **Html5** component should be rewritten
- Other components are deployed as Stateless Deployments
- Resource management should be organized
- Sensitive data like passwords should be securely handled via Secrets

Overall, schematically network communications look like in the Figure 4.3. ConfigMaps and Secrets are skipped intentionally.

Going down to implementation, there is a quite notable moment regarding the deployment of **Redis** and **MongoDB** - there is no need to deploy it manually and we can use out-of-the-box packaging solution by Helm packaging tool. We will come back to this later. Now we start deploying stateless components.

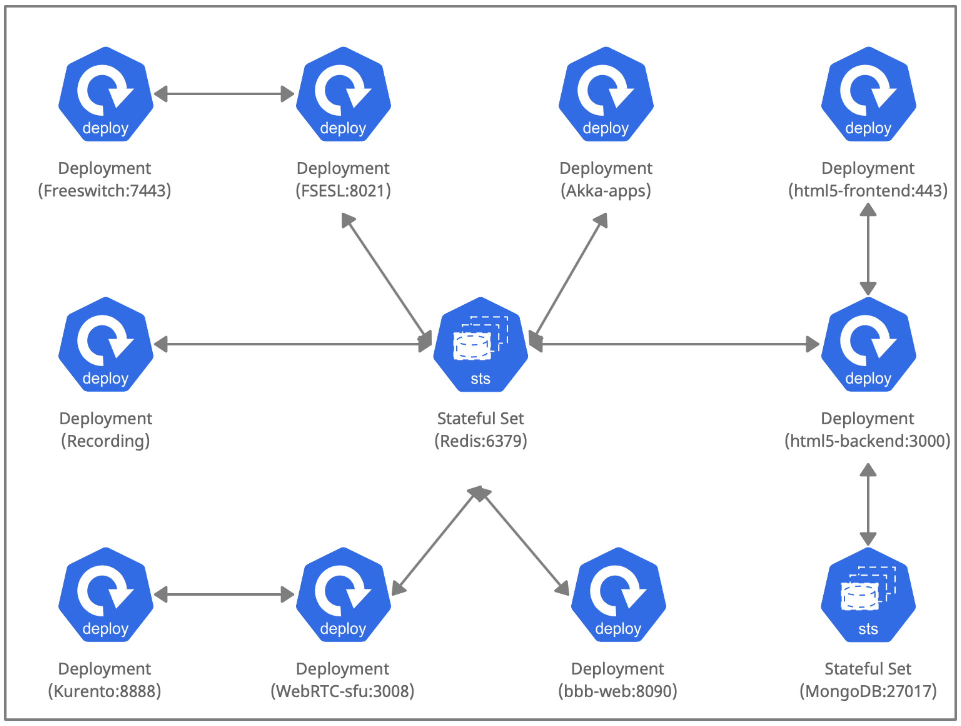

Figure 4.3: BigBlueButton Kubernetes Architecture

- FreeSWITCH

  `freeswitch-deployment.yaml`

```
apiVersion: apps/v1
kind: Deployment # Kubernetes resource kind we are creating
metadata:
  name: freeswitch-app
spec:
  selector:
    matchLabels:
      app: freeswitch-app
  replicas: 1 # Number of replicas that will be created for this
    deployment
  template:
    metadata:
      labels:
        app: freeswitch-app
        environment: development
        tier: backend
        release: canary
        owner: bulatdavlyatshin
    spec:
      containers:
        - name: freeswitch
          image: bulatdavlyatshin/bbb-docker_freeswitch:2.4
          imagePullPolicy: IfNotPresent
          env:
            - name: DOMAIN
              valueFrom:
                configMapKeyRef:
                  name: bbb-config
                  key: DOMAIN
            - name: EXTERNAL_IPv6
              valueFrom:
                configMapKeyRef:
                  name: bbb-config
                  key: EXTERNAL_IPv6
            - name: EXTERNAL_IPv4
              valueFrom:
                configMapKeyRef:
                  name: bbb-config
                  key: EXTERNAL_IPv4
            - name: ESL_PASSWORD
              valueFrom:
                configMapKeyRef:
                  name: bbb-config
                  key: FSESL_PASSWORD
            # - name: SIP_IP_ALLOWLIST
            #   value: "-"
            - name: DISABLE_SOUND_MUTED
              value: "true"
            - name: DISABLE_SOUND_ALONE
              value: "true"
            - name: SOUNDS_LANGUAGE
              value: en-us-callie
          resources: {}
          securityContext:
            capabilities:
              add:
                - IPC_LOCK
                - NET_ADMIN
                - NET_RAW
                - NET_BROADCAST
                - SYS_NICE
                - SYS_RESOURCE
          volumeMounts:
            - mountPath: /Users/bulatdavlyatshin/etc/freeswitch/
    sip_profiles/external
```

```
              name: bbb-volume
            - mountPath: /Users/bulatdavlyatshin/etc/freeswitch/dialplan/public_docker
              name: bbb-volume
            - mountPath: /Users/bulatdavlyatshin/var/freeswitch/meetings
              name: bbb-volume
      volumes:
        - name: bbb-volume
          hostPath:
            path: /Users/bulatdavlyatshin/var/bbb-volume
            type: DirectoryOrCreate
```

`freeswitch-service.yaml`

```
apiVersion: v1
kind: Service
metadata:
  name: freeswitch-service
spec:
  selector:
    app: freeswitch-app
  ports:
    - name: first
      protocol: "TCP"
      port: 8021
      targetPort: 8021
    - name: second
      protocol: TCP
      port: 5066
      targetPort: 5066
    - name: nginx
      protocol: TCP
      port: 7443
      targetPort: 7443
  type: NodePort
```

- Kurento

  `kurento-deployment.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kurento-app
spec:
  selector:
    matchLabels:
      app: kurento-app
  replicas: 1
  template:
    metadata:
      labels:
        app: kurento-app
        environment: development
        tier: backend
        release: canary
        owner: bulatdavlyatshin
    spec:
      containers:
        - name: kurento
          image: kurento/kurento-media-server:6.16
          imagePullPolicy: IfNotPresent
          env:
            - name: KMS_STUN_IP
              valueFrom:
                configMapKeyRef:
                  name: bbb-config
                  key: STUN_IP
            - name: KMS_STUN_PORT
              valueFrom:
                configMapKeyRef:
                  name: bbb-config
                  key: STUN_PORT
            - name: KMS_MIN_PORT
              value: "31000"
            - name: KMS_MAX_PORT
              value: "32768"
            - name: KMS_TURN_URL
              value:
            # - name: KMS_NETWORK_INTERFACES
            #   value: "-"
            - name: GST_DEBUG
              value: 3,Kurento*:4,kms*:4,KurentoWebSocketTransport:5
          volumeMounts:
            - mountPath: /Users/bulatdavlyatshin/var/kurento
              name: bbb-volume
          ports:
            - containerPort: 8888
      volumes:
        - name: bbb-volume
          hostPath:
            path: /Users/bulatdavlyatshin/var/bbb-volume
            type: DirectoryOrCreate

```

`kurento-service.yaml`

```
apiVersion: v1
kind: Service
metadata:
  name: kurento-service
spec:
  selector:
    app: kurento-app
  ports:
    - protocol: "TCP"
      port: 8888
      targetPort: 8888
  type: NodePort

```

- FSESL

  `fsesl-deployment.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: fsesl-akka-app
spec:
  selector:
    matchLabels:
      app: fsesl-akka-app
  replicas: 1
  template:
    metadata:
      labels:
        app: fsesl-akka-app
        environment: development
        tier: backend
        release: canary
        owner: bulatdavlyatshin
    spec:
      containers:
        - name: fsesl-akka
          image: bulatdavlyatshin/bbb-docker_fsesl-akka:2.3
          imagePullPolicy: IfNotPresent
          env:
            - name: FSESL_PASSWORD
              valueFrom:
                configMapKeyRef:
                  name: bbb-config
                  key: FSESL_PASSWORD
            - name: REDIS_HOST
              value: bbb-redis-headless
          ports:
            - containerPort: 8021

```

- WebRTC-sfu

  `webrtc-deployment.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: bbb-webrtc-sfu-app
spec:
  selector:
    matchLabels:
      app: bbb-webrtc-sfu-app
  replicas: 1
  template:
    metadata:
      labels:
        app: bbb-webrtc-sfu-app
        environment: development
        tier: backend
        release: canary
        owner: bulatdavlyatshin
    spec:
      containers:
        - name: bbb-webrtc-sfu
          env:
            - name: CLIENT_HOST
              value: "0.0.0.0"
            - name: KURENTO_NAME
              value: kurento
            - name: REDIS_HOST
              value: bbb-redis-headless
            - name: FREESWITCH_IP
              value: freeswitch-service
            - name: FREESWITCH_SIP_IP
              valueFrom:
                configMapKeyRef:
                  name: bbb-config
                  key: EXTERNAL_IPv4
            - name: EXTERNAL_IPv4
              valueFrom:
                configMapKeyRef:
                  name: bbb-config
                  key: EXTERNAL_IPv4
            - name: ESL_IP
              value: host.docker.internal
            - name: ESL_PASSWORD
              valueFrom:
                configMapKeyRef:
                  name: bbb-config
                  key: FSESL_PASSWORD
            - name: LOG_LEVEL
              value: info
            - name: NODE_CONFIG
              value: '{"kurento":[{"ip":"159.65.138.253","url":"
    ws://kurento-service:8888/kurento"}]}'
          image: bulatdavlyatshin/bbb-docker_webrtc-sfu:2.4
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 3008

```

4.5 Cluster topology for stateful components

Well-known and popular applications that exist in helm repository are easy to deploy with a Cluster or Master-Slave topology.

The Redis Helm chart or the Redis Cluster Helm chart are the two methods available for deploying a Redis Cluster[12]. Both options offer a quick and dependable way to operate Redis in a real-world setting.

- When using the Redis Cluster Helm chart, a cluster of six nodes with several writing points (three masters) and three slave nodes is configured by default. By default, the Redis Helm chart deploys three nodes: one writing point (one master), two replica nodes, and (slaves).

- While the Redis Cluster will create a master-slave cluster using Redis Sentinel, the Redis Cluster Helm chart will deploy a Redis Cluster topology with sharding. Redis allows many databases, unlike the Redis Cluster, which only supports one database and is advised if you have a large dataset. Redirection must be supported by the Redis Cluster client, but not by the Redis client itself.

- The Redis Cluster Helm chart's architecture enables users to access the cluster both internally and externally, and you may scale the cluster up or down in either access.
- Disaster recovery and failover are an extra functionality of the Redis Cluster Helm chart. In the event that the master node or even all the nodes go down, the cluster will automatically recover and new master nodes will be promoted in order to keep the cluster balanced and guarantee uninterrupted read/write operations.

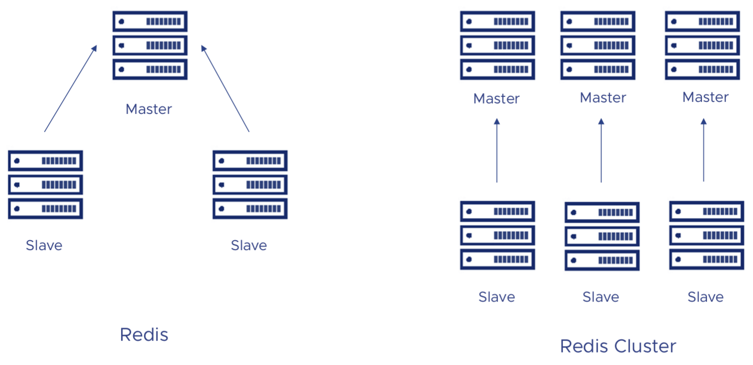

Figure 4.4: Redis Cluster vs. Master-Slave Topology

So the most reliable and stable, yet the most expensive way is choosing a Cluster topology in such scenarios.

Chapter 5

Conclusions and Future Work

The main objective for this thesis work was to understand how we can refactor legacy stateful applications to fit on-premises Kubernetes for orchestrating and better scaling.

To achieve this goal we first installed Kubernetes on our so-called "bare metals". The Kubernetes itself by the way, was installed in a Cluster Topology, i.e. 3 master nodes by 3 worker nodes for high reliability and stability.

After that we analyzed the BigBlueButton architecture and pointed out several important moments such as:

- separating stateless and stateful components
- discovering a service that needs loosening
- sensitive data is not secured
- great demand in resource management

The next challenges we face to make the application Kubernetes-friendly are:

- factor out the front-end part from html5 service
- secure sensitive data with secrets and Vault
- configure resource management, particularly the automatic scaling
- add monitoring tools with turned on notifications

Bibliography

[1] Kevin Casey. «How to explain Kubernetes in plain English». In: 20 (Sept. 2020). URL: https://enterprisersproject.com/article/2017/10/how-%20explain-kubernetes-plain-english (cit. on p. 4).

[2] Gaetano BUSCEMA. «Security orchestration in Kubernetes with Verefoo». MA thesis. Torino: Politecnico di Torino, 2021 (cit. on p. 6).

[3] *Docker Containers Changed How We Deploy Software*. 2019. URL: https://www.magalix.com/blog/kubernetes-101-%20concepts-and-why-it-matters (cit. on p. 6).

[4] Matthew O'Riordan. *Everything You Need To Know About Publish/Subscribe*. URL: https://ably.com/topic/pub-sub (cit. on p. 15).

[5] *CloudVSPrem*. URL: https://www.cleo.com/blog/knowledge-base-on-premise-vs-cloud (cit. on p. 19).

[6] Kelsey Hightower. *Kubernetes the hard way*. URL: https://github.com/kelseyhightower/kubernetes-the-hard-way (cit. on p. 20).

[7] *K8s Core Concepts*. URL: https://www.linkedin.com/learning/advanced-kubernetes-1-core-concepts (cit. on p. 20).

[8] *What is CNI?* URL: https://rancher.com/docs/rancher/v2.x/%20en/faq/networking/cni-providers/ (cit. on p. 34).

[9] *Kubernetes Specifications: StatefulSet*. URL: https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/ (cit. on p. 40).

[10] *Mongo Standalone*. URL: https://kubernetes.io/blog/2017/01/running-mongodb-on-kubernetes-with-statefulsets/ (cit. on p. 41).

[11] *BigBlueButton doc*. URL: https://docs.bigbluebutton.org/2.4/architecture.html (cit. on p. 42).

[12] *Bitnami topologies*. URL: https://docs.bitnami.com/kubernetes/infrastructure/redis/get-started/compare-solutions/ (cit. on p. 50).