2020

High Level Design

POLLINATE - PLATFORM ENGINEERING TECH ASSIGNMENT – V3.0

MEHAR CHAUHAN

Table of Contents

Introduction

This project is designed to demonstrate the ability to record API calls in a highly available, multi-datacenter database. There is much debate about using Kafka topics as a database, but there is good reason for using it in this project, at least for now.

To resolve some of the issues of ensuring that the Kafka cluster is highly available (with the added ability elasticity), the application and Kafka cluster is hosted inside Kubernetes.

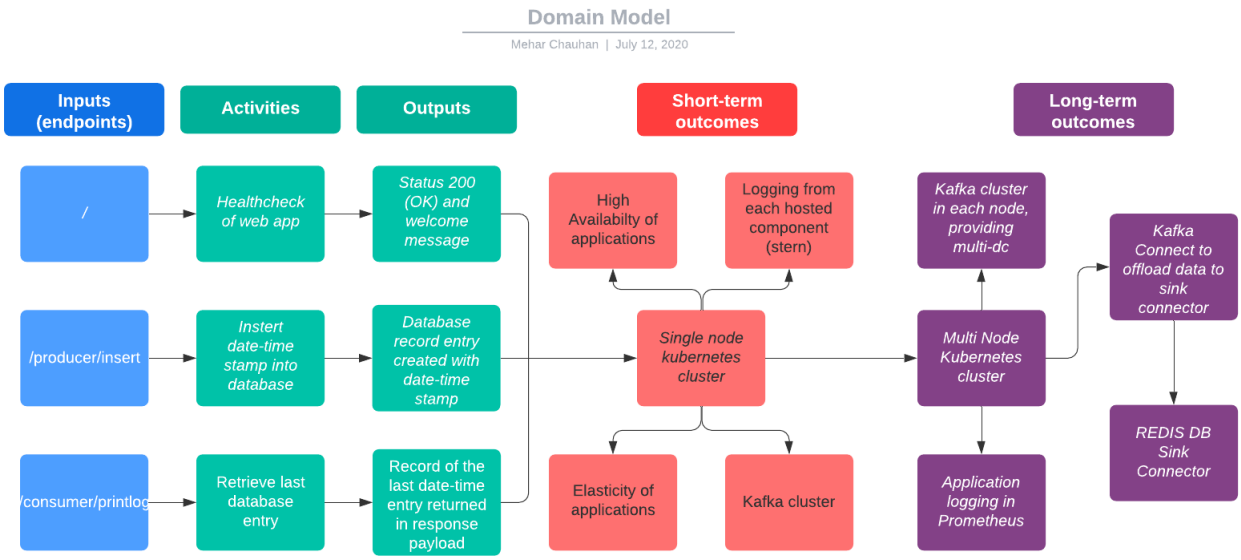
Problem Statement

Build a highly available service that records a date-time stamp into a database. The database itself should also be highly available with multiple datacenters to ensure replication high availability of the data.

Domain Model

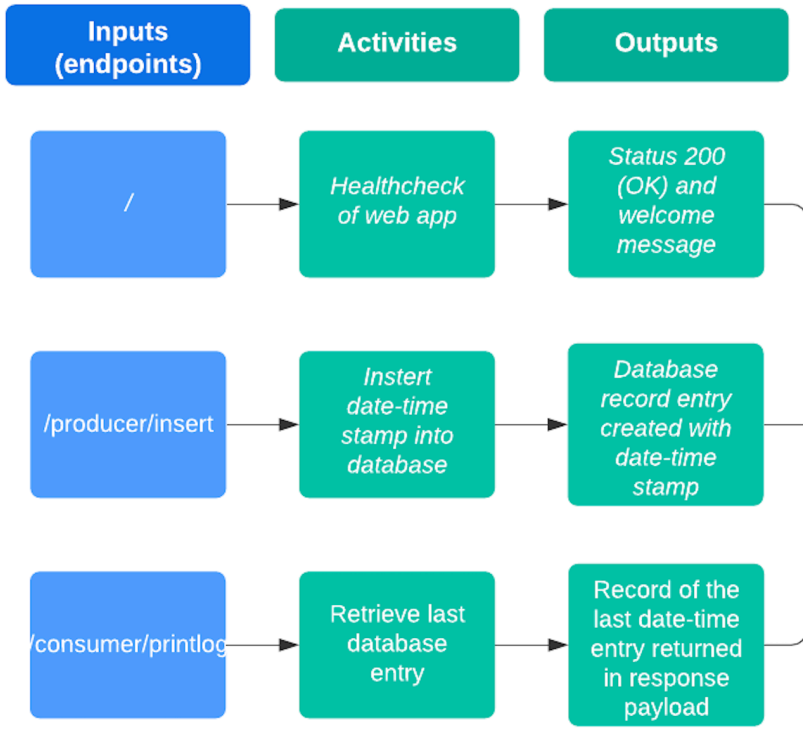
Taking on board the requirements, I have constructed a Domain Model, that covers both the short and long-term outcomes, expected to be delivered from this project.

The short-term explicitly covers the deliverables for this POC (see below)



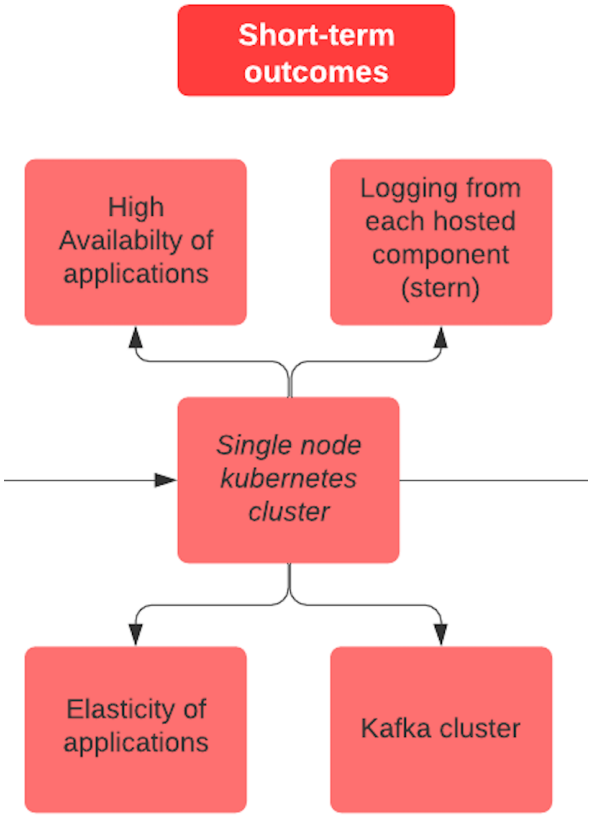
I have broken this out into 3 sub-headings below:

Inputs / Outputs and Activities



For simplicity of the Web Application, there will be 3 endpoints.

Short-term Outcomes



The web application along with the Kafka cluster will be hosted on a Kubernetes (minikube) environment. The choice of minikube was purely time-based and satisfied the short-term requirement.

With the Kubernetes cluster (be it minikube), the following is provided as vanilla:

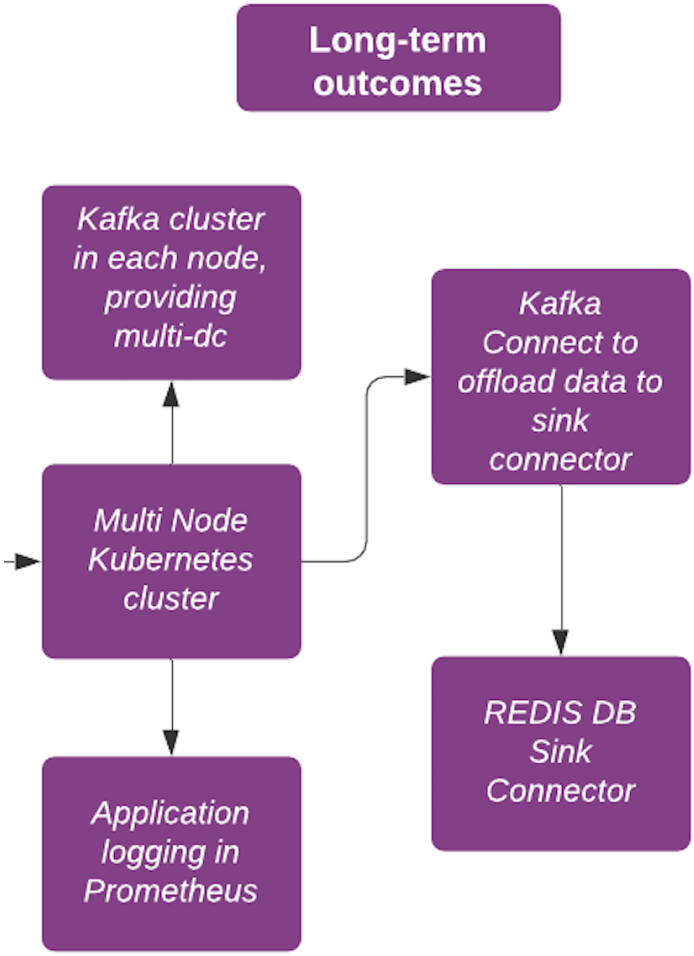
- high availability of deployed components provided a `deployment` resource is provisioned with an attached `replicaset`
- elasticity / scalability of components is available to the Kubernetes administrator:
    - e.g. create high-availability of the web application can be achieved by issuing the command:

```
$ kubectl scale --repliacs=2 deployment/<application name>
```

Kubernetes allows one to interrogate application logs (generated by `stdout` and `stderr` to enable a developer/administrator to debug. There are additional tools that'll allow one to quickly debug multiple components across the cluster, such as the tool: stern[1]

[1] stern - GitHub project

Long-term Outcomes



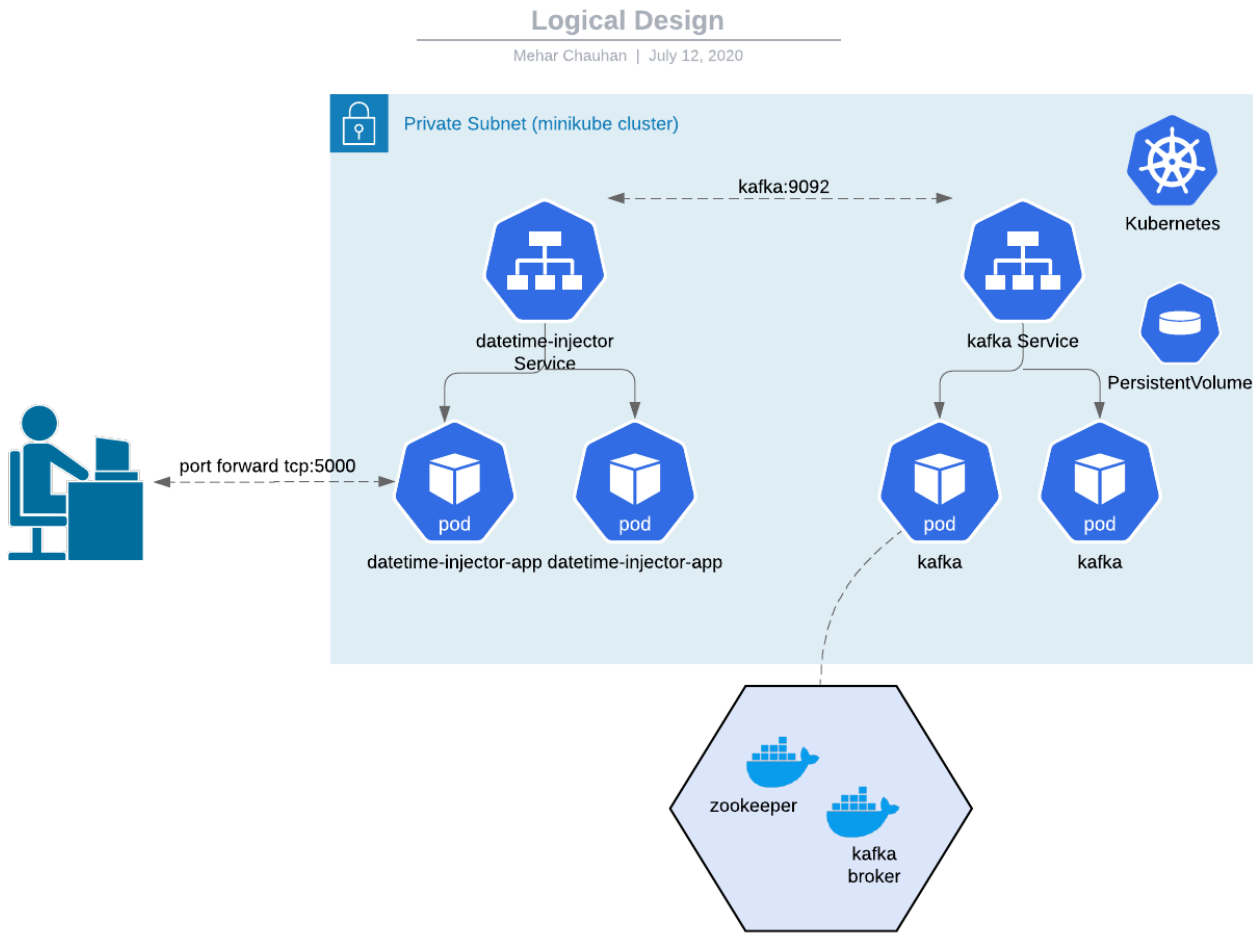
Should this project have further longevity beyond a POC, I would provision a multi-node Kubernetes cluster, deploying a Kafka cluster on each worker node, to create a multi-datacenter.

Following on from multi-dc, I would be looking into data replication through the means of Confluent's Kafka replicator, or the community licensed Apache Kafka's

If the requirement arose to offload the data into an external database, perhaps a data lake, I'd be considering Kafka Connect (specifically a Kafka Sink Connector).

Furthermore, to transform the data, Kafka Steams and KSQL (to filter)

Logical Design



This design illustrates the minikube cluster along with how the deployed HA components are load-balanced by a `service` resource.

The Kafka service itself is a statefulset with a PVC, the diagram depicts just the `persistentvolume` to keep things simple.

The web application itself will only be accessible by port-forwarding using the command:

```
$ kubectl port-forward <web application pod> 5000
```

1. Kafka messages are placed on a topic called `datetime-topic` when the actor makes a `POST` request to the `datetime-injector-app` web application.
2. The message is placed and persisted on that Kafka topic indefinitely

Design Decisions

Why Choose Kafka Topics as a Database

I chose Kafka topics as they hold the same capabilities of a database, such as conforming to ACID (Atomicity, Consistency, Isolation and Durability).

The data held on a Kafka topic is held indefinitely and the data can be queried (using KSQL) should the requirement arise.

The Kafka cluster can do more than store records on a database, such as transformation of data, streaming, and can offload data from topics to databases such as REDIS, mySql. There is flexibility and extensibility with this approach.

Future Considerations

Monitoring

Kubernetes provides various logging capability out of the box, this can feed into a data analytics tool such as Prometheus or Grafana

API Spec (Swagger)

There are various ways to publish an API specification. One intuitive way of achieving this is by generating a swagger document. It holds several benefits, such as:

1. being a live document (updated as and when the API changes)
2. can mock the service endpoints to test the response payloads