Adidas Challenge

Ezequiel Molinero
emolinero@gmail.com
April 2019

Overview

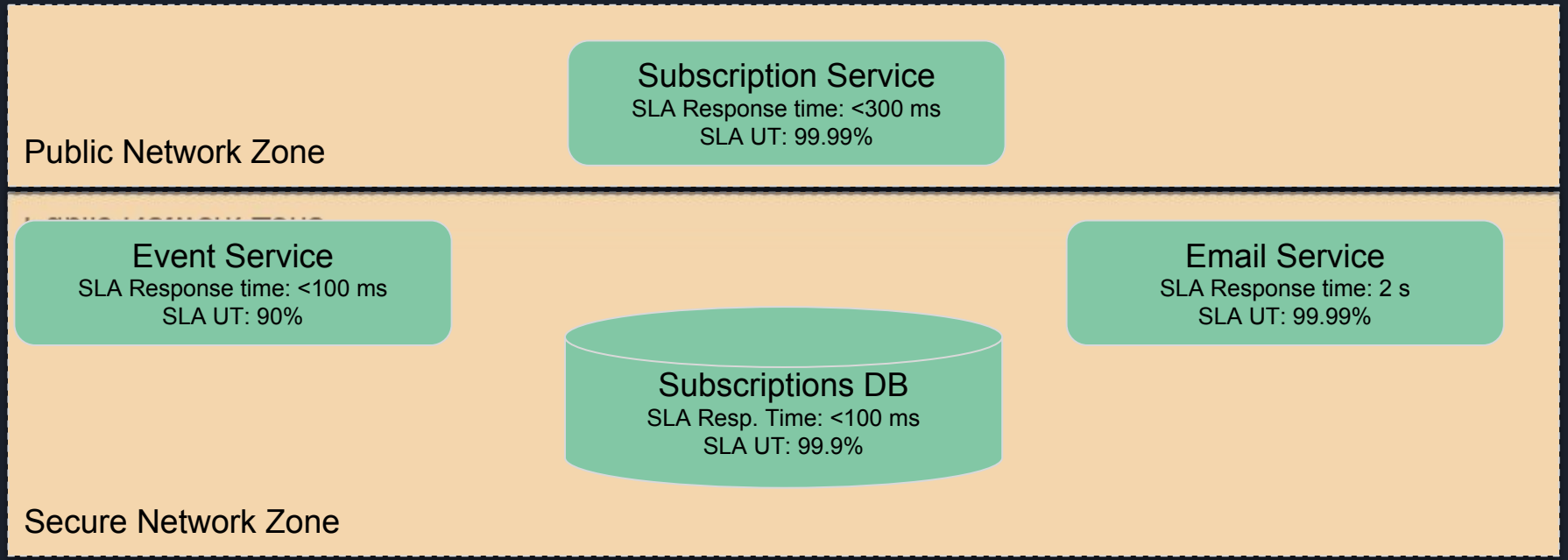
According to the exercise given I developed a solution to match the specified requirement

I tried to fulfil the requirements with my better knowledge and understanding

Although I have been working in different architecture disciplines as Enterprise Architecture or Functional Architecture I could put the exercice to work.

Some technical specifications or NFR were considered in overall solution as theoretical without a practical implementation

Required Architecture Landscape



Implementation

- Four modules have been created: subscriptionService, eventService, emailService and jmsService (the latest a helper)
- Using spring-boot framework, maven, docker and several auxiliary classes.
- *subscriptionService* as the exposed application, receives POSTs request as if they were coming from a Web Application customer facing
- It stores the subscription with Customer and Newsletter information, along with the flag of consent
- Then it keeps listening on a queue for event messages
- *eventService* receives POSTs request of events, related to a newsletter in particular
- As a new event appears, it send it to events queue
-

Implementation

- *subscriptionService*, upon the event, searches for subscriptions matching the event received
- Sends the needed email messages to another queue
- *emailService* is litening on that other queue, and sends mails according to that message
- *jmsService* is present to create a broker for managing queues
- For sending POST requests to subscription service a JWT token is needed. That represents the web application authentication
- For security on other layers it would be recommended application similar methods: such as private/public key sharing between services for obtaining tokens
- A Dockerfile is created for each service, all of them are built based on a base JAva machine (for details of each building process please refer to README.md)

Usage

- Authentication

  http://localhost:8080/login?username=user&password=123456

- Subscription Creation

  POST -> localhost:8080/subscriptions?Authorization Bearer=<JWT TOKEN>

```
{
        "email": "prueba@prueba.com",
         "firstName": "Prueba",
         "gender": "M",
         "dateOfBirth": "2019-04-02T18:28:34.273Z",
         "consent": true,
         "newsletterId": "1"
}
```

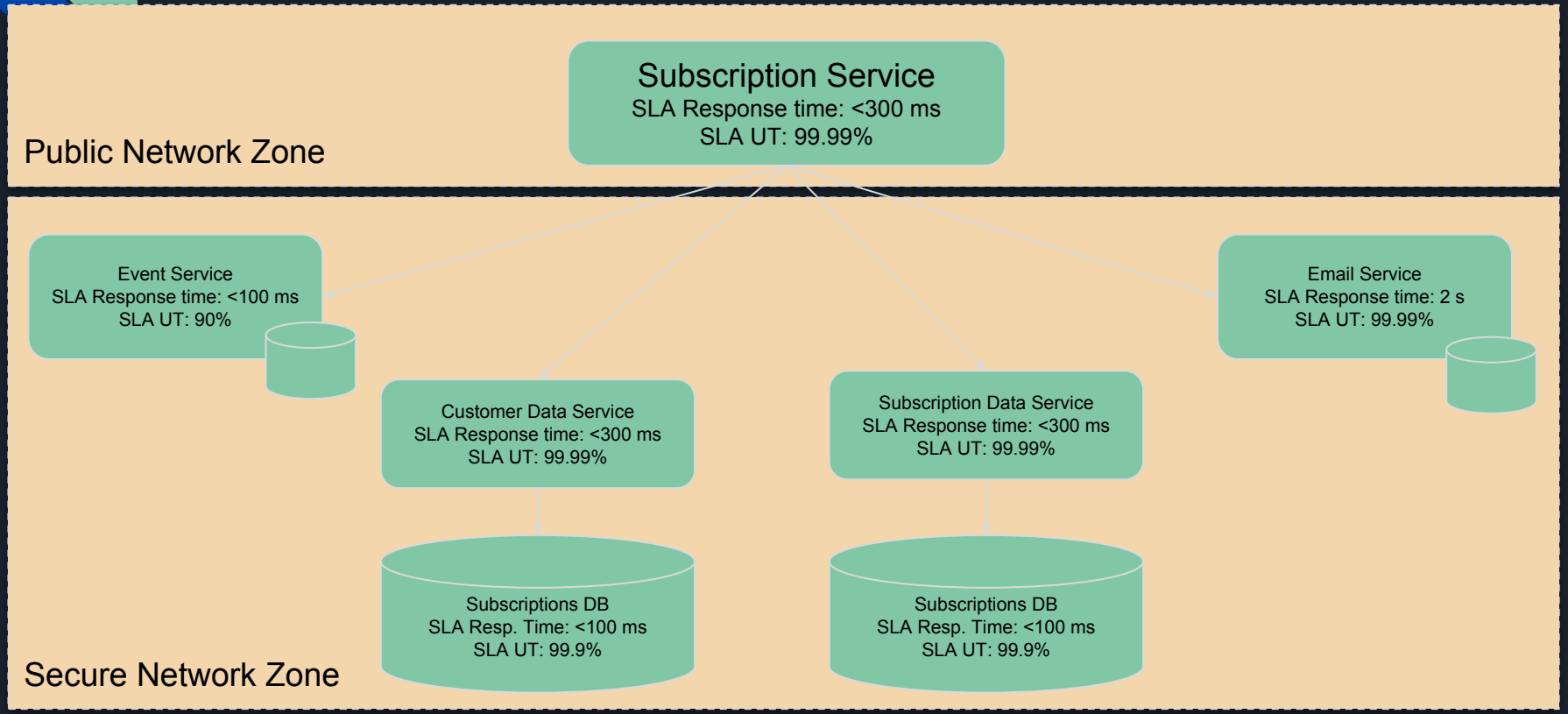
Usage

- Event Creation

  POST -> localhost:8090/event

```
{
        "description": "Campaign Newsletter 1",
        "newsletterId": "1"
}
```

Proposed Architecture Landscape



Security

- *subscriptionService* offers a token (jwt) authorization to control who can execute its endpoints

- Networking - Physical

There is none implemented, but the network segregation would create a host control to access services. Where it can be controlled with IP addresses, ports, and even application layer control

SLA Conditions

Based on needed SLA conditions I suggest, during Testing a complete Performance & Stress Test. To be able to get threshold times and availability factors.

Once in a theoretical production environment I suggest alarms on Monitoring applications, such as Prometheus.

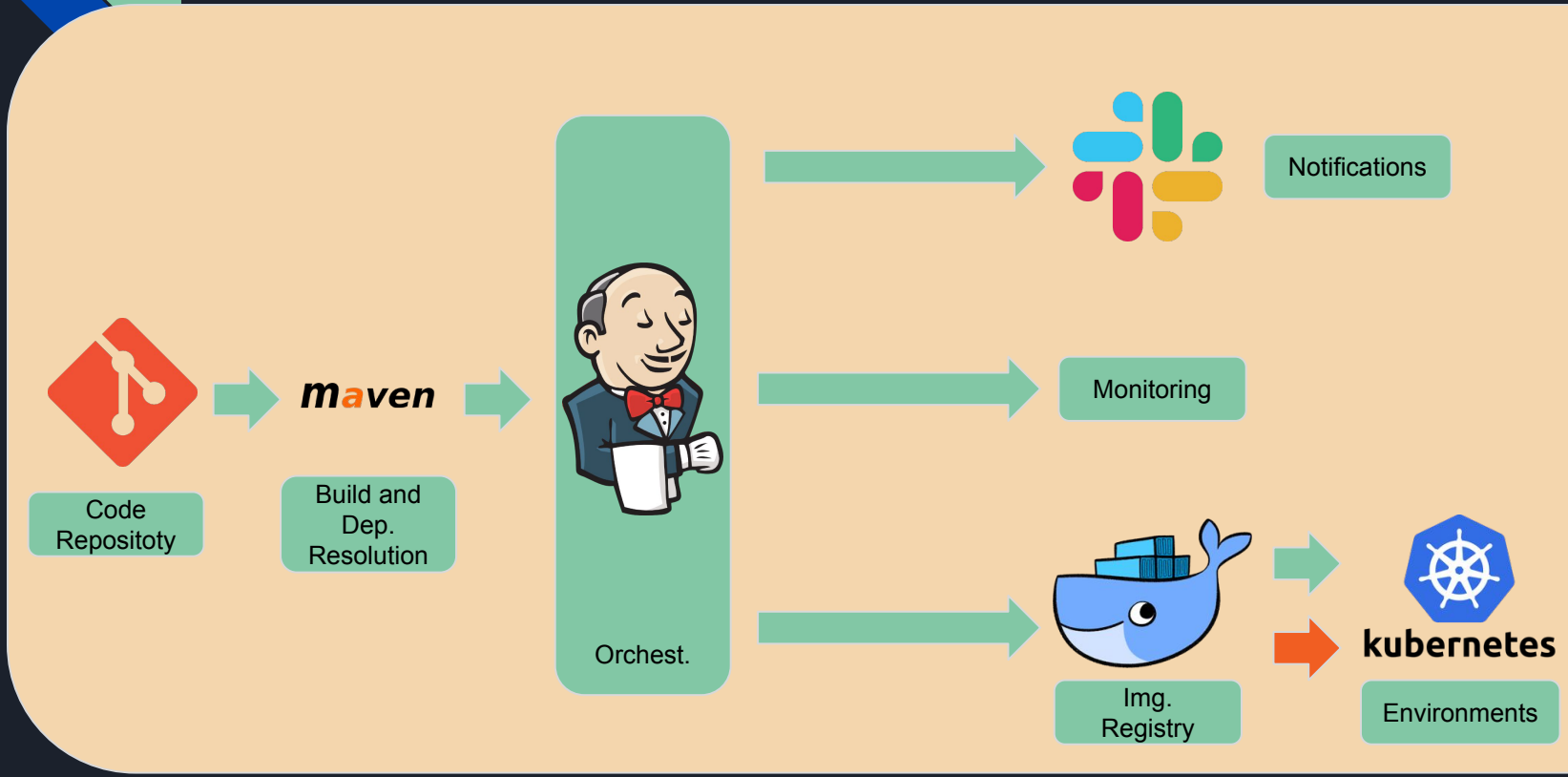
The application itself should integrate with Kubernetes when in risk of losing SLA conditions in order to activate new instances

Suggestions

Within the known ecosystem I suggest:

- The creation of a Customers Service
- The modeling of the nature of events in campaigns, products related, or whatever defines them. There I can extend and expand the customizing of email templates and make the customer engagement, better.
- The use of Cloud Services for email delivery (after email service) not to overcharge company infrastructure and IP with email sending.
- The creation of a Subscriptions Data Service, because of the network jump and not to expose the database. As there I can tighten security aspects
- The use of several layers firewalls to control north-south traffic coming from the DMZ

Proposed Life Cycle



Thank you