CLUMIO

Clumio New Start Guide - 2020

Table of contents

Architectural Style

Clumio as a product is built using Cloud Native architecture. To understand any software system well, it is paramount to understand it's architectural style. At a very high level, if an architecture assumes that the system is going to run ONLY on and depend on an infrastructure provided by a cloud vendor, it is called Cloud Native architecture.

The link points you to a basic definition of what Cloud Native Architecture is. But do your own research and study to gain a deeper understanding, and how and why companies are using it, pros vs cons, etc. Few reading pointers: link 1, link 2

The cloud vendor that is currently supported by Clumio, is AWS.

Layers of the Architecture

At a high-level, there are 4 major layers in this Cloud Native architecture,

1. Cloud Infrastructure
2. System Deployment Definition
3. Service
4. Tools

If you have understood what Cloud Native architecture is, fairly well, you now realize that every component(**Service**) of the system is developed as a microservice and deployed independently. These microservices are deployed on a **Cloud Infrastructure** (AWS in our case) which offers computing infrastructure like CPU, Memory. Storage and also fundamental infrastructure services like S3, DynamoDB, SQS etc. All these microservices are connected using some kind of network interfaces. You may think of the whole deployment as a graph with nodes and edges, let's call the graph **System Deployment Definition**.

Ex: **microservice1** and **microservice2** are 2 nodes and they are connected using a bidirectional edge, **HTTP**. This means that microservice1 and microservice2 can talk to each other over HTTP.

Apart from the edge details above, we also need to describe the computing resources (CPU, MEMORY, # of instances etc) required on **Cloud Infrastructure (**AWS) for each microservice. If you extend this to the above example, **microservice1 (2CPU and 2GB memory)** is connected over **HTTP** to **microservice2 (3CPU and 4GB memory).** Apart from basic resources like CPU/Memory, other infrastructural components like S3, DynamoDB that are offered by underlying cloud infrastructure are also considered resources in cloud-native architecture.

If you can visualize, the above nodes and edges along with the definition of each nodes and their edges, this is collectively called **System Deployment Definition** and this may become pretty complex for a large deployment. To define, maintain and manage this complex **System Deployment Definition,** we use a tool called Terraform. At this point, we have looked at 2 layers. **Cloud Infrastructure & System Deployment Definition.**

Now that we have defined our service's infrastructure needs, let us try to understand what **Service**s themselves are made of.

Each **Service** fundamentally is a running computer program that can

1. Receive a request,
2. Process the request
3. Send the response back[optional]

To achieve above, Service authors can choose any programming language that fits the use-case the Service is solving. Only contract that the Service has to stick to is it will honor the request, response interfaces [Edges in the graph analogy] it defines and publishes to other Services in the system.

At Clumio, we use Golang as a programming language to develop Services. An exception is the UI service, whose purpose is to render and serve HTML UI pages. We use ReactJS as a programming language in the case of UI Service.

The Service Interfaces [Edges in the graph analogy] can be built on top of many protocols. We use the following protocols primarily at Clumio and it is important to understand how these work.

1. HTTP - UI from Webbrowser talks to Clumio services only via HTTP. We follow REST API architecture here.
2. GRPC - All services(except UI Service) talk to each other using GRPC and uses protobuf as data transfer format. Used in case of synchronous communication.
3. SQS - An event-based queuing protocol used for Asynchronous communication.

Make sure to get familiar with HTTP, REST-API, gRPC, Protobuf, SQS if you are not familiar with them already.

Every service (code) is built as a docker container, and runs on AWS EKS. Reading materials on docker and AWS EKS available later in this doc.

All 3 layers we talked about so far, are used in the runtime of the system. But before the system gets into a running state, there are several steps involved and we use many **Tools** for each of them. **Tools** are a side and cross-cutting layer in the system.

A noncomprehensive list of Steps involved outside runtime and tools used for each.

1. Code Version System - Github
2. Build the code and create necessary deployment artifacts - Jenkins
3. Host and maintain deployment artifacts - Dockerhub
4. Create and maintain System Deployment Definition - Terraform.io
5. Deploy the code built as per System Deployment Definition - Jenkins
6. Quality Assurance automation of the deployed code - This are a bunch of automated test cases written in python and other technologies defined under the Automation section in the document later.
7. Running of the Quality Assurance automation - Jenkins

Materials to Read

The following are some links that point to specifics and good reading materials on the technologies we talked about above so far. You are expected to go through these and get yourself familiar with concepts.

Deployment

- **Docker :** https://www.docker.com/why-docker
- **Kubernetes:** https://www.youtube.com/watch?v=4ht22ReBjno
- **AWS Services:** AWS EKS, SQS, DynamoDB, API Gateway
- AWS official documentation is good. Do google.
- **Terraform**: https://www.terraform.io/intro/index.html

Golang

1. **Why golang** - https://letzgro.net/blog/9-reasons-to-choose-golang-for-your-next-web-application/
2. **Concurrency is not Parallelism**: https://talks.golang.org/2012/waza.slide#1
3. **Coding practice** - https://gophercises.com/ -
4. https://medium.com/@eminetto/clean-architecture-using-golang-b63587aa5e3f

CLUMIO

UI Technologies (NOTE: Try out the tutorials yourself while watching the videos)

1. **Importance of UI**
   https://blog.omni-bridge.com/5-reasons-why-ui-is-important-75a65408c330
2. **Javascript**
   Tutorial - https://www.youtube.com/watch?v=PkZNo7MFNFg
   - Event Loop: https://www.youtube.com/watch?v=8aGhZQkoFbQ
3. **Typescript** - https://www.typescriptlang.org/
4. **CSS** and **Sass**
   Tutorial - https://www.youtube.com/watch?v=1Rs2ND1ryYc
   - Responsive Web Design - https://www.youtube.com/watch?v=srvUrASNj0s
   - Sass - https://sass-lang.com/documentation
5. **Single Page App** : https://en.wikipedia.org/wiki/Single-page_application
   - SPA vs MPA/MVC: https://www.youtube.com/watch?v=F_BYg2QGsC0
6. **React** Tutorial - http://bit.ly/reactTutorial
7. **Redux** https://redux.js.org/
8. **Enzyme** https://airbnb.io/enzyme/

Automation

1. **PyTest Framework** - https://docs.pytest.org/en/latest/
   a. Additional reading on writing tests with the PyTest framework - https://medium.com/tenable-techblog/automation-testing-with-pytest-444c8b34ead2
   b. Tutorial video - https://www.youtube.com/watch?v=bbp_849-RZ4
2. **API Automation**
   a. Introduction to REST API automation with Python and PyTest - https://medium.com/@peter.jp.xie/rest-api-testing-using-python-751022c364b8
   b. Tutorial videos - https://www.youtube.com/playlist?list=PLIMhDiITmNrILoYaVsrxwteH6LqMr07uX
   c. Postman
3. **UI Automation**
   a. Selenium WebDriver for UI automation
      i. Tutorial video - https://www.youtube.com/watch?v=oM-yAjUGO-E
      ii. Documentation - https://selenium-python.readthedocs.io/
   b. Splinter (Selenium wrapper library) -
      i. Tutorial video - https://www.youtube.com/watch?v=ApA7EVwSzg0
      ii. Documentation - https://splinter.readthedocs.io/en/latest/
   c. Selenium Grid
      i. Tutorial video - https://www.youtube.com/watch?v=kAvzKA9wsbo

   ii. Documentation - https://selenium.dev/documentation/en/grid/
d. Zalenium (dynamic test resource provisioning and orchestration)
   i. Tutorial video - https://www.youtube.com/watch?v=_C5bekbxNpc
   ii. Documentation - https://opensource.zalando.com/zalenium/

4. **Continuous Integration and Development**
   a. Jenkins CI/CD - https://jenkins.io/doc/
   b. Jenkins pipelines (jenkinsfiles) - https://jenkins.io/doc/book/pipeline/