2020

CSCI 49378: Lecture 8: Cloud Systems and Infrastructures II

Bonan Liu
CUNY Hunter College

NYC Tech-in-Residence Corps

Cloud Systems and Infrastructures II

Bonan Liu
Tech-In-Residence Member, Hunter College, CUNY
Spring 2020

Disclaimer

The content of this presentation is being provided for educational and informational purposes only. The views, thoughts, and opinions expressed in this presentation belong solely to the author, and not necessarily to the author's employer.

The content of this presentation is not endorsed by the author's employer.

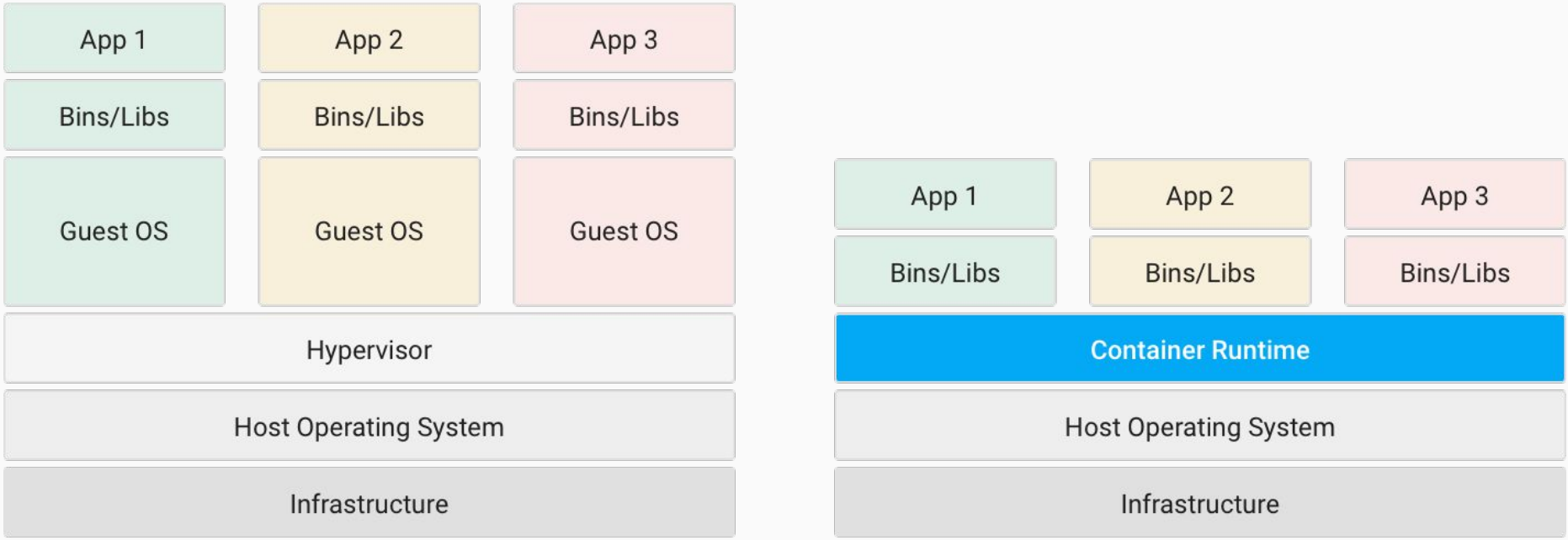
Agenda

- Containers
- Cloud Functions/Run
- API and Communication

Containers

Containers is an abstraction of the application packing and it allows the application to decouple with the running environment.

Containers


App 1
App 2
App 3
Bins/Libs
Bins/Libs
Bins/Libs
Guest OS
Guest OS
Guest OS
Hypervisor
Host Operating System
Infrastructure
Virtual Machines
App 1
App 2
App 3
Bins/Libs
Bins/Libs
Bins/Libs
Container Runtime
Host Operating System
Infrastructure
Containers

Containers

Containers benefits:

- Easy deployment / low operational burden
- Isolation between applications
- Less resource usage for small applications
- Easy migration
- Multi-Cloud / Hybrid-Cloud Support

Kubernetes Components:

- Master Components: Control Plane
  - Control Plane vs Data Plane
- Node Components: Running Environment
- Addons: Monitoring and Tooling

Containers

Kubernetes Concepts:

- Images
- Images Registry
- Nodes
- Pods
- Deployments

Cloud Functions

Cloud Functions is a event-driven computing services for lightweight computational tasks.

Cloud Functions could be very useful in developing event-driven workload such as data pipelines, IoT backends or machine learning data processors.

Small Demo: Create a Cloud Functions function.

API and Communication

Generally, an application programming interface (API) is a set of subroutine definitions, communication protocols, and tools for building software. [1]

- Definitions: written in a configuration language. (yaml, proto3)
- Communication Protocols: HTTP, pubsub, CORBA
- Tools for building software: usually in the form of production-quality SDK

API and Communication

gRPC is an open source RPC framework which offers high-performance configurable API implementation and multi-language support.

- proto as message definition language
- support Streaming and RESTFul API
- rich tooling with multi-language support

Demo: Create a Restful API with gRPC

Practice: Create your own API with gRPC.

To generate gRPC code:

```
python -m pip install grpcio
python -m grpc_tools.protoc -I.
--grpc_python_out=. --python_out=plugins=grpc:.
student.proto
```

API and Communication

Student.proto

```
syntax = "proto3";

// The student profile service definition.
service StudentService {
  // Returns a student resource.
  rpc GetStudent (GetStudentRequest) returns (Student) {}
}

// The request message containing the user's hunter ID.
message GetStudentRequest {
  int32 id = 1;
}

// Student resource message.
message Student {
  int32 id = 1;
  string name = 2;
}
```

API and Communication

client.py

```
import sys
import grpc

import student_pb2
import student_pb2_grpc


def main(id):
  with grpc.insecure_channel('localhost:50051') as channel:
    stub = student_pb2_grpc.StudentServiceStub(channel)
    response = stub.GetStudent(student_pb2.GetStudentRequest(id=id))
    print(response.name)


if __name__ == '__main__':
  main(int(sys.argv[1]))
```

API and Communication

server.py

```
from concurrent import futures
import time
import grpc

import student_pb2
import student_pb2_grpc

_HUNTER_STUDENTS_ = {1:"STU1", 2:"STU2",3:"STU3"}

class StudentService(student_pb2_grpc.StudentServiceServicer):

  def GetStudent(self, request, context):
    return student_pb2.Student(id=request.id, name=_HUNTER_STUDENTS_[request.id])
```

API and Communication

Server.py (continued)

```
def serve():
  server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
  student_pb2_grpc.add_StudentServiceServicer_to_server(StudentService(), server)
  server.add_insecure_port('[::]:50051')
  server.start()
  try:
    while True:
      time.sleep(60*60) # 1 hour
  except KeyboardInterrupt:
    server.stop(0)


if __name__ == '__main__':
  serve()
```

Readings

Borg, Omega, and Kubernetes. Google Research. *https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/44843.pdf*

One of the following two:

- *Google's API design guide. https://cloud.google.com/apis/design/*
- *Microsoft's API design guidance. https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design*