ORACLE

Cloud Native Fundamentals

Prasenjit Sarkar
Oracle Cloud Infrastructure
February 2020

Safe harbor statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.

The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

Objectives

After completing this lesson, you should be able to;

- Have a general idea of Cloud Native Application Fundamentals
- Describe the patterns for building cloud-native apps
- Describe the Fallacies of Distributed Systems
- Describe Cloud-Native Building Blocks

ORACLE

Introduction to Cloud Native

Understanding Cloud-Native

"... natively utilizies service and infrastructure from cloud computing providers"

"... approach to build e& run apps that exploit the advantages of the cloud computing model"

"... describes container-based environments, deployed as microservices and managed on elastic infrastructure through agile DevOps, continuous delivery workflows"

"... build, run, and improve apps based on well-known techniques and technologies for cloud computing"

"... collection of small, independent, and loosely coupled services"

Pets vs. cattle

Pets 🐶

- Treat your infrastructure like pets
- Give them names, IP addresses, Care of them, keep them updated

Cattle 🐮

- Everything is just a number
- No attachment
- If something goes wrong, you replace it

Understanding Cloud-Native

Apart from focusing on business logic, you will realize the following when building cloud native applications for the first time:

- I am dealing with services running across multiple machines
- I am dealing with network and communication between these services

ORACLE

Patterns for building cloud-native apps

Cloud-Native vs. Traditional Architectures

Stateful vs. Stateless

- State stored with the compute instance
- Load balancers using sticky sessions
- What happens on reboot or crash

Service orchestration vs. Service choreography

- Multiple services orchestrated to work as one, using sync communication
- Choreography uses eventing system

Dealing with failures

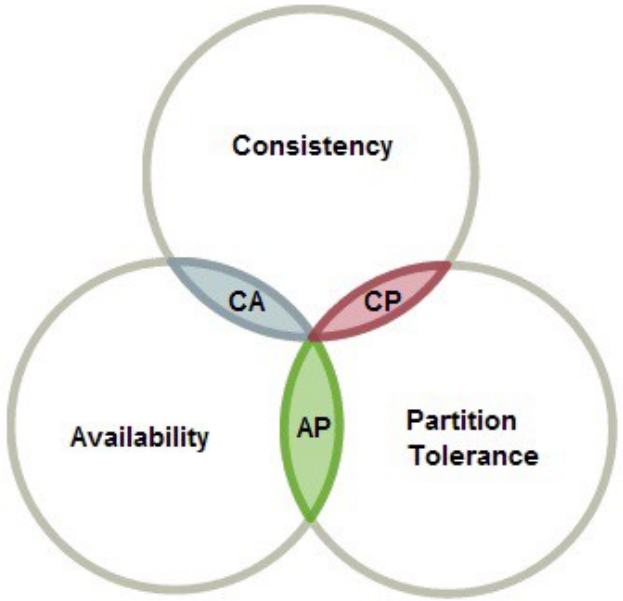
- Minimize failures vs. expect and deal with them

ORACLE

CAP Theorem

CAP Theorem

- Make compromises
- Partitions will always exist
- Optimize for Consistency or Availability



ORACLE

Fallacies of Distributed Systems

8 Fallacies of Distributed Systems

1. Network is reliable
2. Latency is zero
3. Infinite bandwidth
4. Network is secure
5. Topology does not change
6. There is one administrator
7. Transport cost is zero
8. Network is homogeneous

ORACLE

Cloud-Native Building Blocks

Microservices vs. Containers vs. Functions

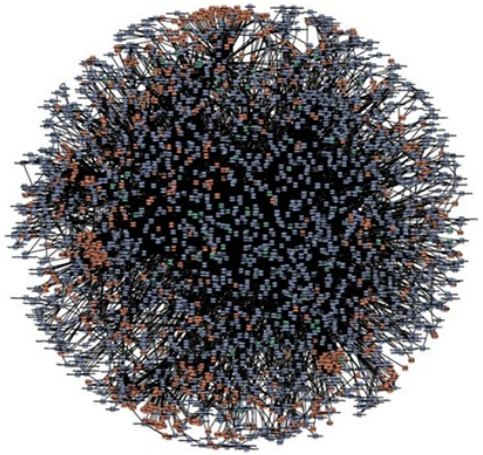
Microservices = architectural style

Functions & Containers = technologies serving a particular purpose

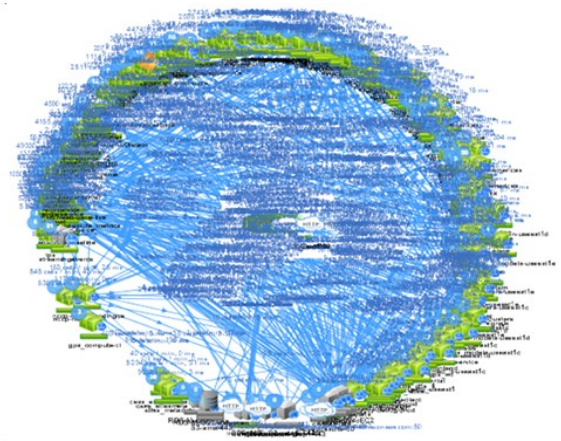
Understand how to best use functions & containers, together with eventing/messaging technologies to design, develop and operate cloud-native microservices-based applications

Microservices

- Service-oriented architecture
- Loosely coupled services
- Organized around business capability



amazon.com



NETFLIX

Microservices

- Smaller code bases
- Managed by independent teams
- Independently deployable
- Single, well-defined task
- Communication through APIs
- Own tests, builds, data, deployments

Benefits

- Fast(er) verification, deployment, and releases
- Easier to deliver new value
- Use the best tools/frameworks/languages for the job
- Move quicker, faster ramp up time, focus on smaller piece
- One rotten apple won't "poison" other apples
- Able to scale services at di"erent rates
- Easier to measure and observe individual services, specific functionality

Challenges

- Complexity - fallacies of distributed systems
- Decentralized data makes transactions difficult
- Performance - network adds overhead
- Lack of tools for development and testing
- Versioning, backward and forward compatibility
- Inconsistent naming, types, values, etc. when logging and monitoring
- Service dependency management
- Service availability

Security

*9 out of 10 cybersecurity professionals are troubled by cloud security issues (esp. data loss & breaches)

However.

Cloud environments are safer than most on-premises environments

BUT

That doesn't mean you can ignore the security

Defense-in-depth approach (1/2)

Source code

- Secure (private) repository (track and audit access)
- Vulnerability checks as part of the continuous integration

Container image

- Image contains the bare-minimum needed

Container registry

- Use private registry (track and audit access)
- Image vulnerability scanning (e.g. Twistlock)

Defense-in-depth approach (2/2)

Pods

- Images pulled from approved registries only
- Use pod security policies to control volumes, priviledged containers, host ports, networking, etc.

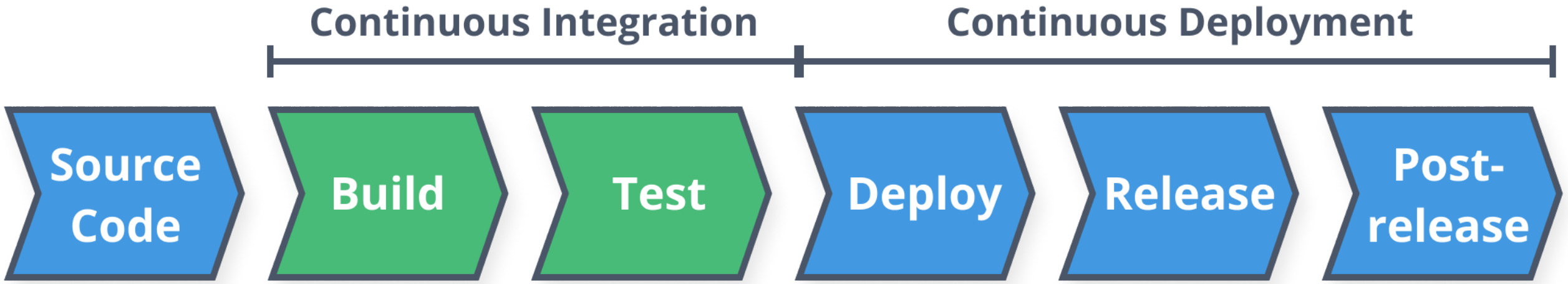
Cluster/Orchestrator

- Secure access to the cluster
- Enable RBAC (Role-based access control)
- Enable audit logs

ORACLE

Operating Cloud-Native Applications

CI/CD



CI - Source Code Control

- Repository where your code lives
- Source of truth for your code/configuration
- Branching strategies
- Mono or multi-repo

Branching Strategies

Git Flow Strategy

- Designed around releases
- Start with master and develop branch
- Use feature, release, hotfix branches
- Helps with feature/release/hotfix tracking

Once release is complete:

- Merge to develop
- Merge to master + tag with a version

CI - Build and Test Stage

Build the code

Run the tests

- Unit/Component
- Linters
- Static analysis

Version/tag the generated artifact

- Use Git commit checksum hash + build number
- Result is versioned artifact*

***Docker image or serverless function package**

CD - Deploy Stage

- No source code beyond this point
  - Images, packaged artifacts, config/deploy templates
- Automatically triggered by successful CI
- Prepare everything needed and place the artifact into staging
- Run tests (canary) & monitor the services

CD - Release Stage

You need enough data from previous stages to feel comfortable releasing the service into production

- Swap stage & production deployment slots
- Redirect a % of the production workload to deployed services

Monitor and observe released versions

- Integrate with alarm system

Rollback to previous version OR keep increasing to 100% (fix-forward)

- Usually done manually
- Could be automated

CD - Post-Release Stage

Part of testing in production/operating the application

- Continous service monitoring
- Investigating incidents/errors
- Alerting/monitoring systems
- Doing chaos testing

Monitoring and Observability

Monitoring is used to assess and report on the overall health of a system or services using metrics

Error rate

- Rate of failing requests (e.g. HTTP 500)
- Incoming request rate
- How much request is coming into the system (HTTP requests/second)

Latency

Time it took to process a request

Utilization

Usage of different pieces of the system (e.g. CPU, memory, disk usage)

ORACLE

Oracle Cloud always free tier:
oracle.com/cloud/free/

OCI training and certification:
cloud.oracle.com/en_US/iaas/training
cloud.oracle.com/en_US/iaas/training/certification
education.oracle.com/oracle-certification-path/pFamily_647

OCI hands-on labs:
ocitraining.qloudable.com/provider/oracle

Oracle learning library videos on YouTube:
youtube.com/user/OracleLearning

Thank you