

ORACLE®

Cloud Native Labs



Paths to Cloud Native

Choose Your Own Adventure

Jesse Butler Principal Cloud Native Advocate, OCI. @jlb13

#OracleCloudNative
cloudnative.oracle.com

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, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

The Old World

- Once upon a time, proprietary systems and software were bundled and sold as a unit
- This created independent silos per vendor, each with ecosystems of tools and service vendors
- Systems analysts surfaced system data and implemented improvements



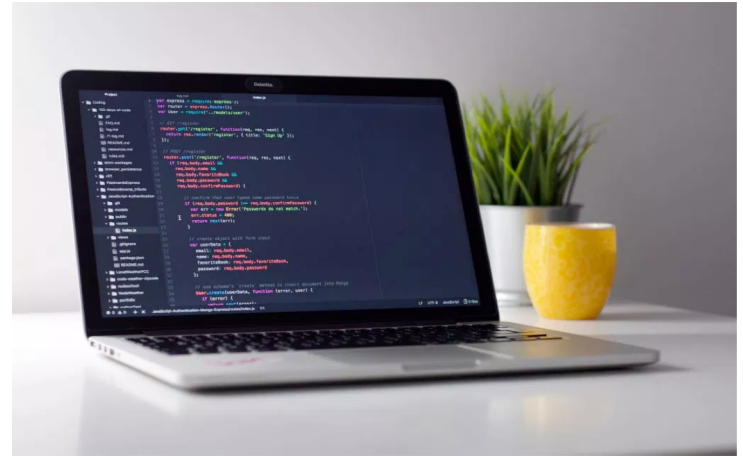
More Recent History

- There were a lot of moving parts in the typical Old World IT organization
- The advent of web applications made time to market a far more sensitive metric
- DevOps arose as a means of reducing friction between where software is created and where it is deployed



Advent of DevOps

- DevOps brings the concerns of development and operations closer together, much like systems analysts
- Developers think in terms of deployments, dev and ops work closely to ensure quality
- DevOps is as much a cultural shift as it is technical



DevOps, Mother of Invention

- Microservices
- Continuous Integration
- Continuous Delivery
- Containers
- Cloud Adoption



Getting to the Cloud, Two Ways

- Coming to the cloud at this point can be daunting, but fear not!
- Happily, this is proven ground with a couple of viable paths to entry
- Largely dependent upon team dynamics and organizational culture



ORACLE®

Cloud Native Labs



Containers and Kubernetes

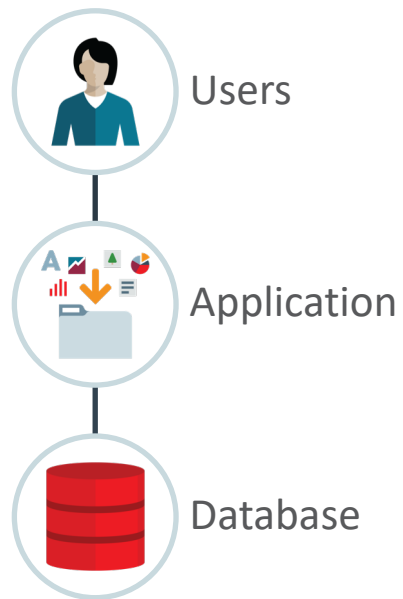
Path One, Tried and True

#OracleCloudNative
cloudnative.oracle.com

Monolithic Applications



Monolithic Applications



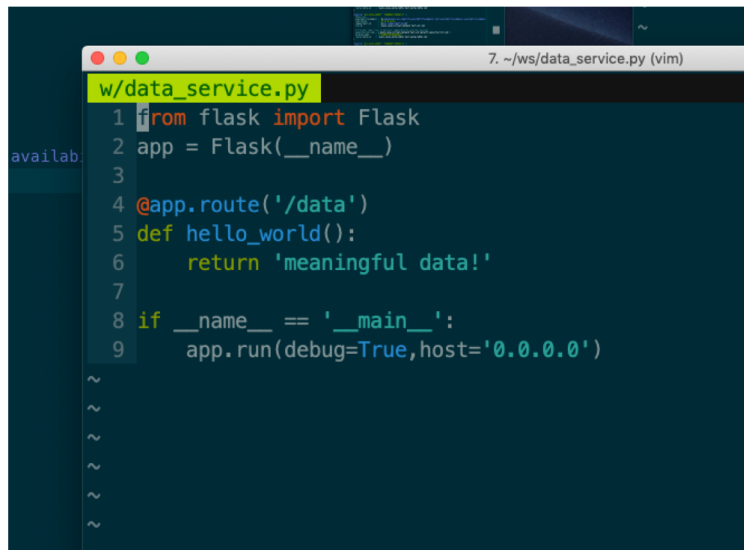
Microservices

- Microservices are the de facto standard for cloud native software
- Microservices allow development teams to deploy portable and scalable applications
- Microservices can be difficult to manage and monitor, putting burden on Ops and DevOps.



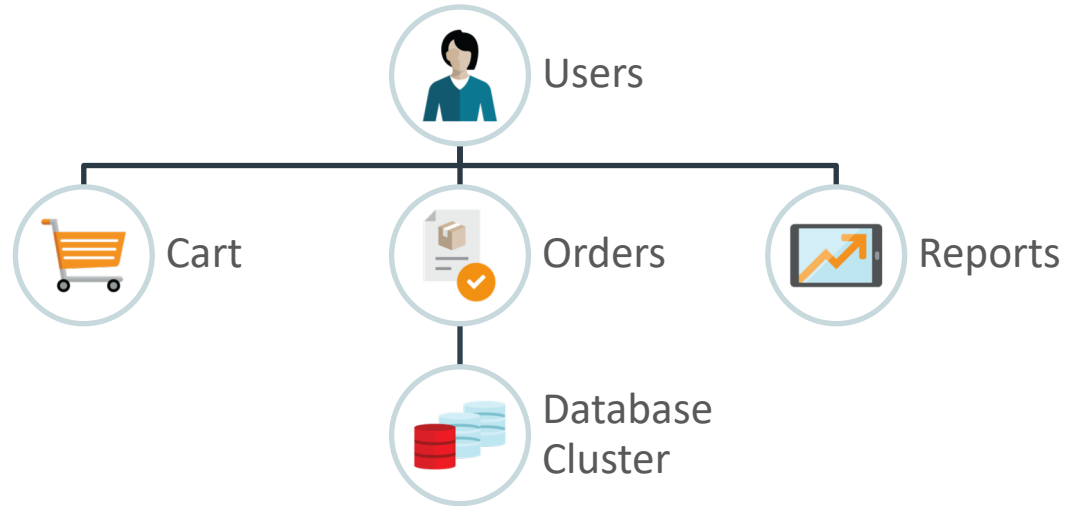
Microservices Example

- Microservices should do one thing, promoting separation of concerns
- Microservices should be idempotent and stateless

A screenshot of a code editor window titled "w/data_service.py (vim)". The code is written in Python and uses the Flask web framework. It defines a single route for the path "/data" which returns the string "meaningful data!". The application is configured to run on host "0.0.0.0" with debug mode enabled. The code is as follows:

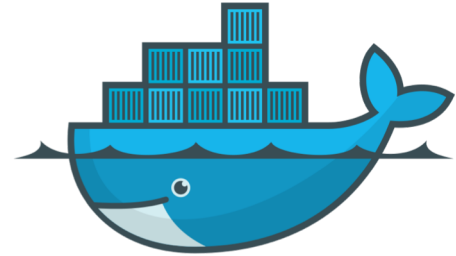
```
w/data_service.py
1 from flask import Flask
2 app = Flask(__name__)
3
4 @app.route('/data')
5 def hello_world():
6     return 'meaningful data!'
7
8 if __name__ == '__main__':
9     app.run(debug=True, host='0.0.0.0')
```


Microservices



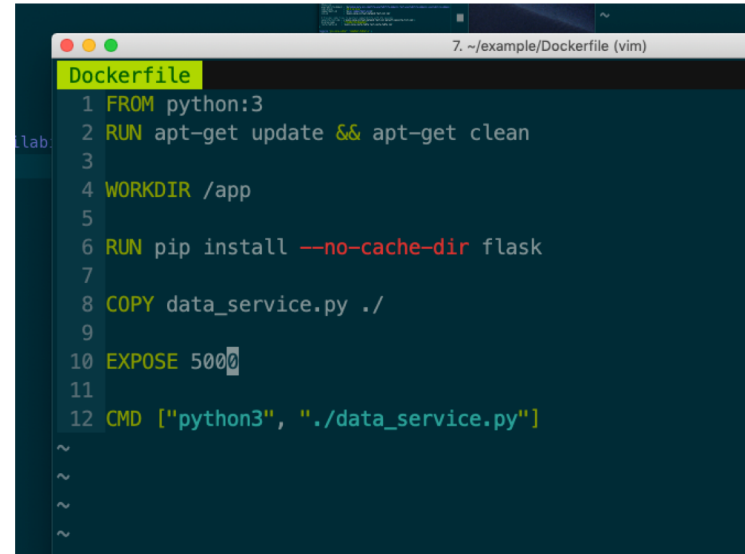
Docker

- Docker changed the way we build and ship software
- Application and host are decoupled, making application services portable
- Containers are an implementation detail, but a critical one



Docker Example

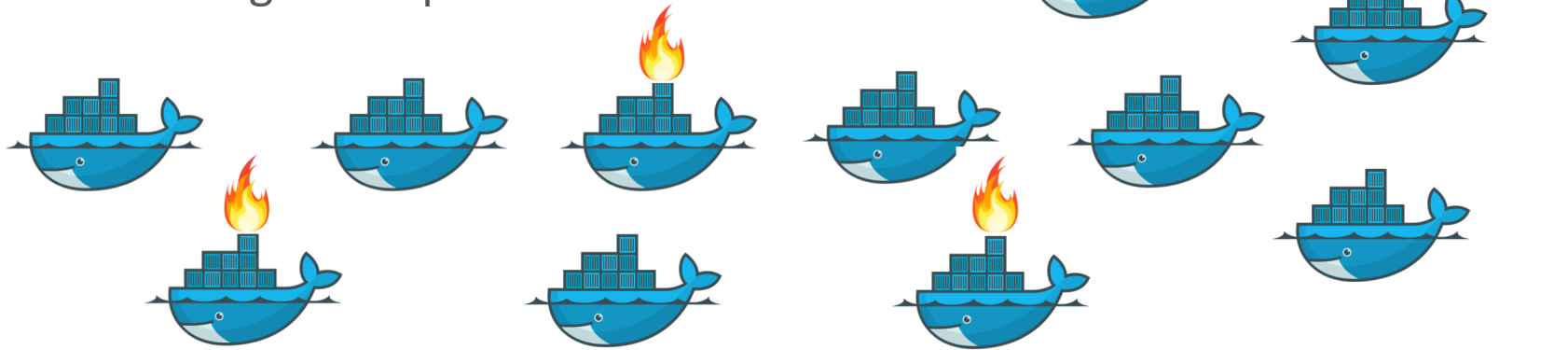
- Docker is used in production at massive scale every day
- Interactively, a development utility for creating containers and container images
- Dockerfile defines content of a container and its runtime configuration
- 'docker build. -tag data_service:1.0'



```
7. ~/example/Dockerfile (vim)
Dockerfile
1 FROM python:3
2 RUN apt-get update && apt-get clean
3
4 WORKDIR /app
5
6 RUN pip install --no-cache-dir flask
7
8 COPY data_service.py ./
9
10 EXPOSE 5000
11
12 CMD ["python3", "./data_service.py"]
```

Docker Is a Start

But, once we abstract the host away by using containers, we no longer have our hands on an organized platform.



Kubernetes

Kubernetes provides abstractions for
deploying software in containers at scale

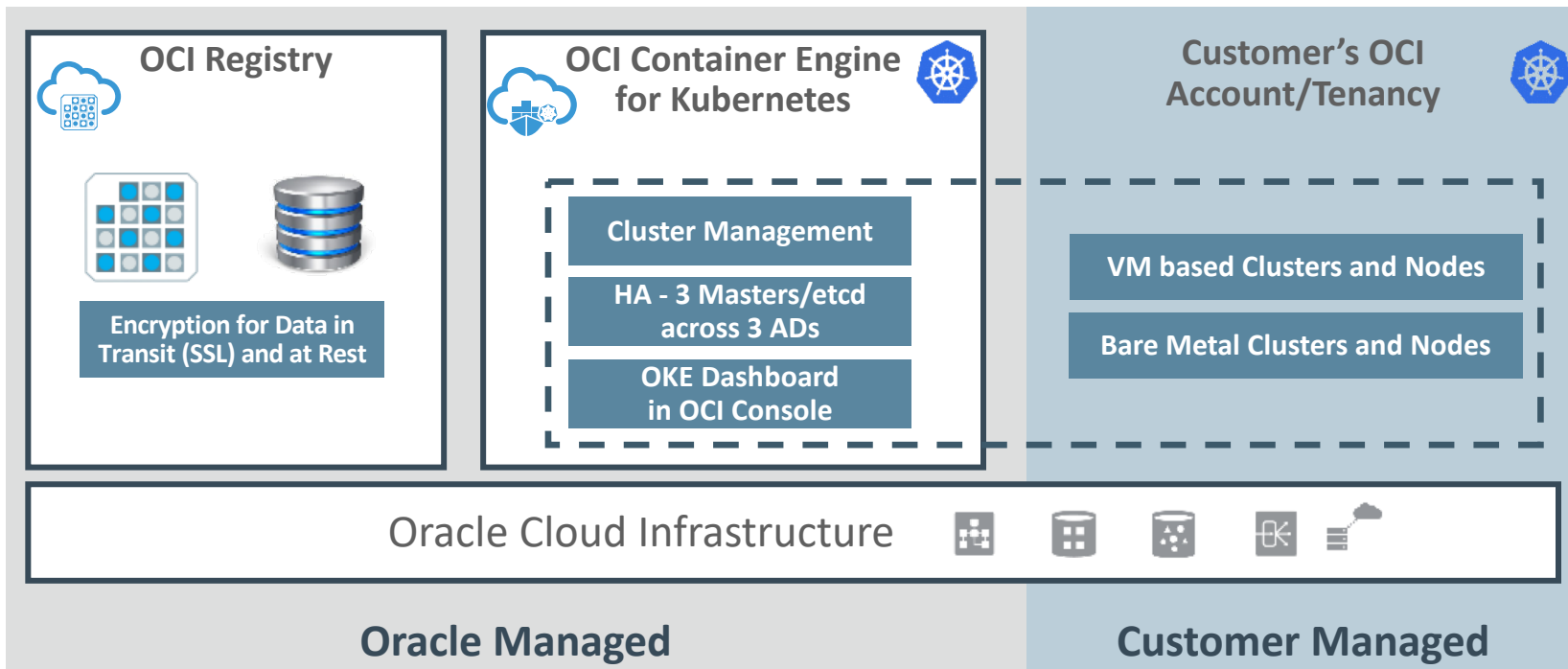
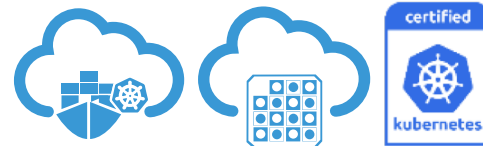


Kubernetes as a Platform

- Infrastructure resource abstraction
- Cluster software where one or more masters control worker nodes
- Scheduler deploys work to the nodes
- Work is deployed in groups of containers



Working with OKE and OCIR on OCI



Kubernetes Example

- Deployments are defined in YAML
- We define what images to use to create our containers, configuration elements, how many instances to run
- Kubernetes makes it happen, and keeps it all running as defined
- 'kubectl create -f' and glory awaits

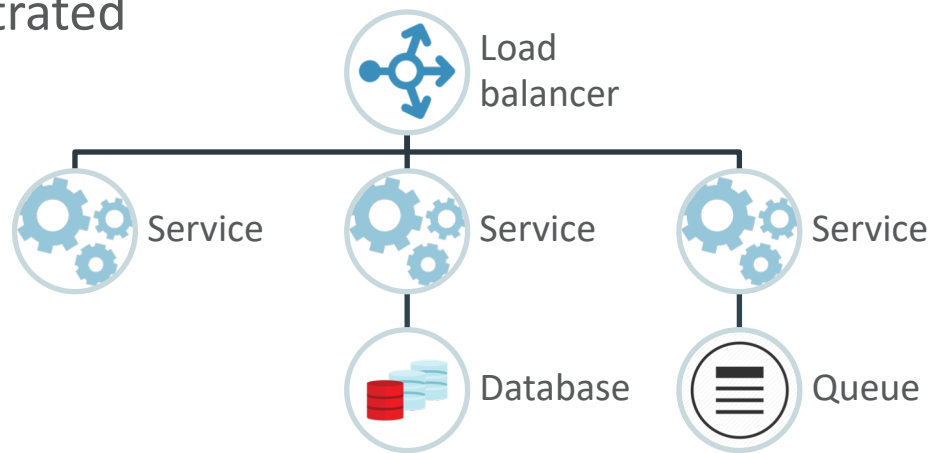
```
deployment.yaml
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: data_service
5 spec:
6   selector:
7     matchLabels:
8       app: data_service
9   replicas: 1
10  template:
11    metadata:
12      labels:
13        app: data_service
14    spec:
15      containers:
16        - name: data_service
17          image: jim-bob/data_service:1.0
18          ports:
19            - containerPort: 5000
20      ---
21  apiVersion: v1
```


Migration from the Old World...



...to Cloud Native Kubernetes Hotness

- Microservices running in orchestrated containers
- Everybody's happy
- What happens now?



Day Two

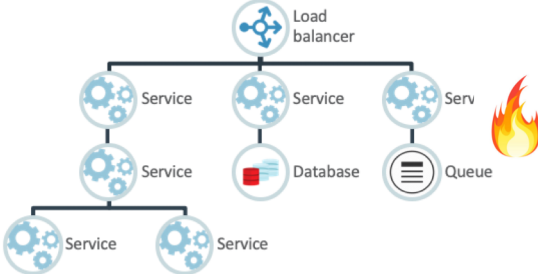
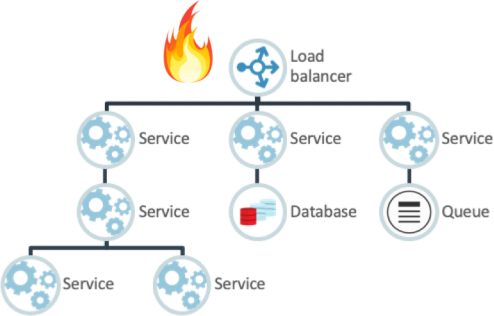
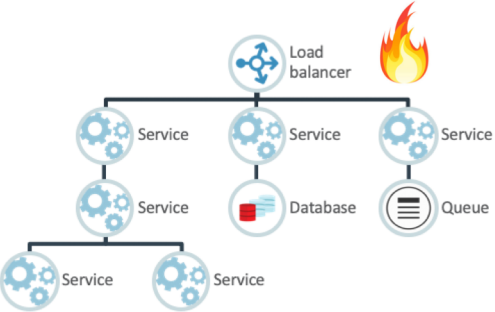
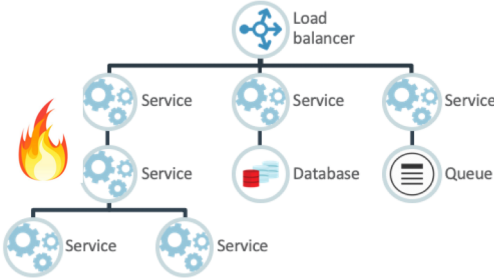
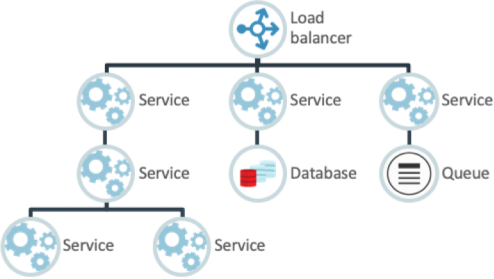
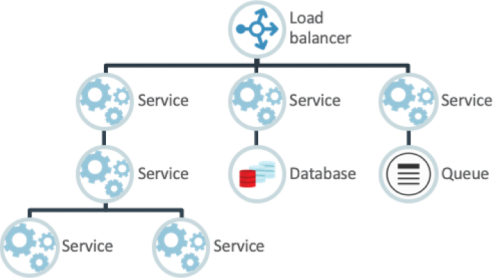
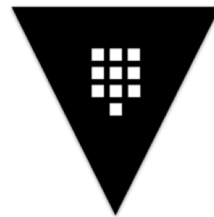


Table Stakes for Services at Cloud Scale

- We require a method to simply and repeatably deploy software, and simply and recoverably modify deployments
- We require telemetry, observability, and diagnosability for our software if we hope to run at cloud scale

Beyond Basics

- Ingress and Traffic Management
- Tracing and Observability
- Metrics and Analytics
- Identity and Security



API Gateway

- Exposes backend services as an API, acting as a reverse proxy
- Routes all calls as needed through the gateway to the appropriate service
- Single point of entry for AuthN, encryption and tracing



Service Mesh

- Infrastructure layer for controlling and monitoring service-to-service traffic
- A data plane deployed alongside application services, and a control plane used to manage the mesh
- Transparent to deployments, leveraging a proxy in the data plane in order to inject services



Continuous Integration / Continuous Deployment

- Continuous integration is the practice of developers integrating iterative progress
- Continuous deployment is a release strategy which deploys the latest tested versions of software available
- Automation is critical for these practices in production



More Containers, More Problems

- Docker and Kubernetes gets us rolling
- Solutions exist for what we need from there, service mesh makes it tidy
- Still, there are more decisions to make...



Cloud Native Computing Foundation

- Container orchestration ecosystem is constantly evolving
- CNCF supports the ecosystem and fosters community
- Began with Google's donation of Kubernetes, now over 30 projects



**CLOUD NATIVE
COMPUTING FOUNDATION**

This landscape is intended as a map through the previously uncharted terrain of cloud native technologies. There are many routes to deploying a cloud native application, with CNCF Projects representing a particularly well-traveled path.

The landscape is organized into several main sections:

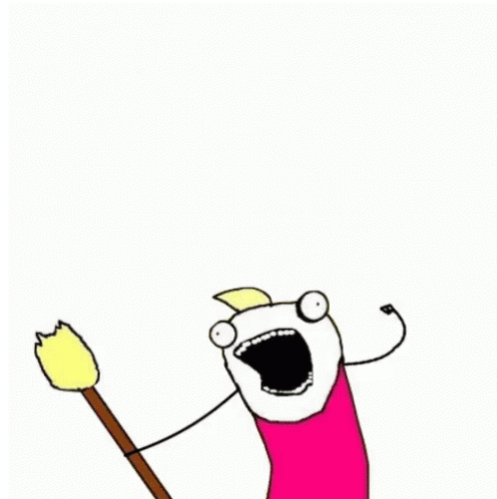
- App Definition and Development:** Includes Database, Streaming & Messaging, Application Definition & Image Build, and Continuous Integration & Delivery.
- Orchestration & Management:** Includes Scheduling & Orchestration, Coordination & Service Discovery, Remote Procedure Call, Service Proxy, API Gateway, and Service Mesh.
- Runtime:** Includes Cloud-Native Storage, Container Runtime, and Cloud-Native Network.
- Provisioning:** Includes Automation & Configuration, Container Registry, Security & Compliance, and Key Management.
- Cloud:** Includes Public and Special categories.
- Platform:** Includes Certified Kubernetes - Distribution, Certified Kubernetes - Hosted, Certified Kubernetes - Installer, and PaaS/Container Service.
- Observability and Analysis:** Includes Monitoring, Logging, Tracing, and Chaos Engineering.
- Serverless:** A dedicated section for serverless technologies.
- Kubernetes Certified Service Provider and Kubernetes Training Partner:** Lists various providers and training partners.

Key logos and projects visible include: Vitess, K8s, Helm, Jenkins, ArgoCD, Istio, Prometheus, Grafana, FluentD, and many others.



Microservices: Cloud Native Path 1, Tried and True

- Microservices in containers on Kubernetes in the cloud is the new compute paradigm
- It's not magic, it's not simple, but it is proven and learnable and doable
- Large and expanding ecosystem and community, great learning materials, helpful people, and plenty of room in the pool
- Of course, there's another choice...



ORACLE®

Cloud Native Labs



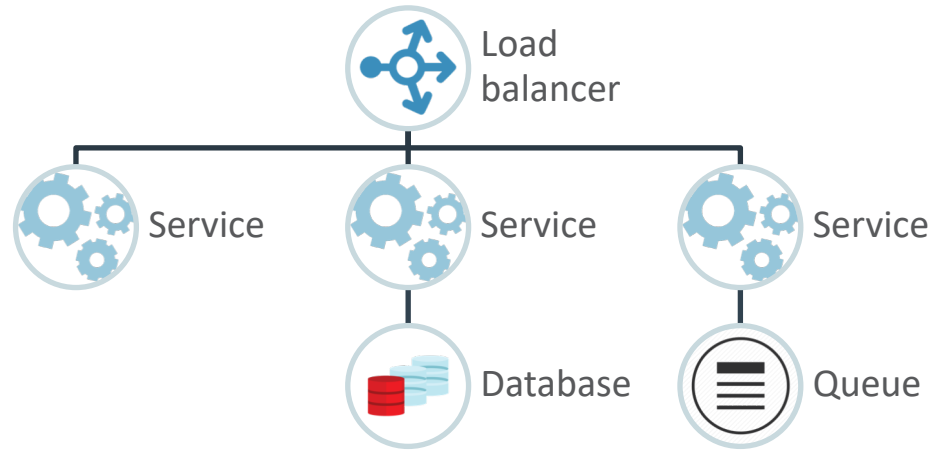
Serverless

Path Two, Brave and Rewarding

#OracleCloudNative
cloudnative.oracle.com

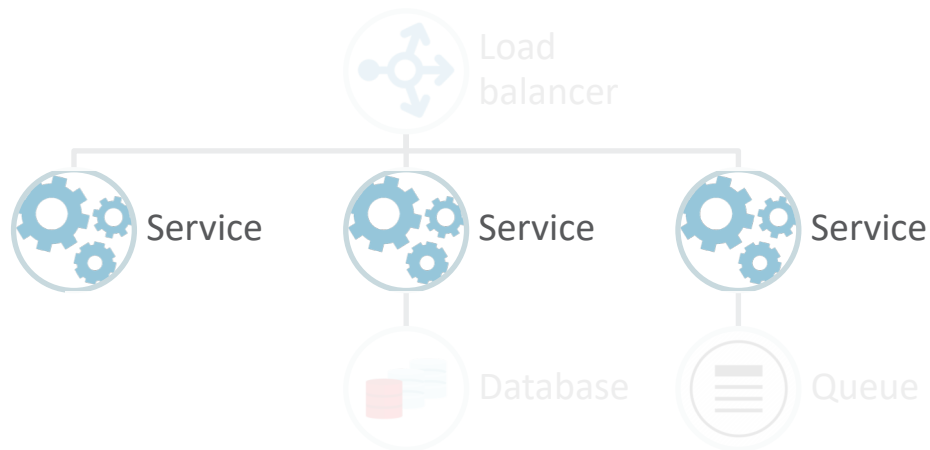
Microservices

Deploying Code to Systems We Build in the Cloud with Excellent Abstractions



Serverless

Deploying Code to Systems We Build in the Cloud with Excellent Abstractions

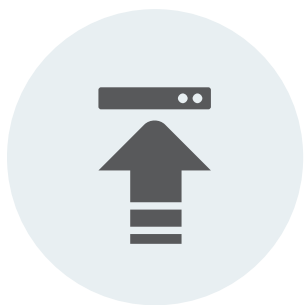


What Is Serverless?

- Event-driven architecture
- Invisible infrastructure
- Automatic scaling on demand
- Granular billing for execution only
- Fault tolerant and highly available



How Does it Work?



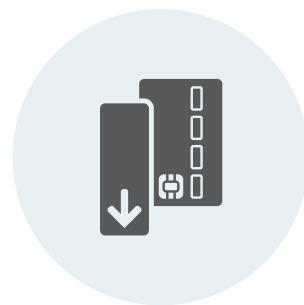
Upload
Function
Source Code



Configure
Function
Trigger



Function is
invoked
when
triggered



Billed for
execution time,
not idle time

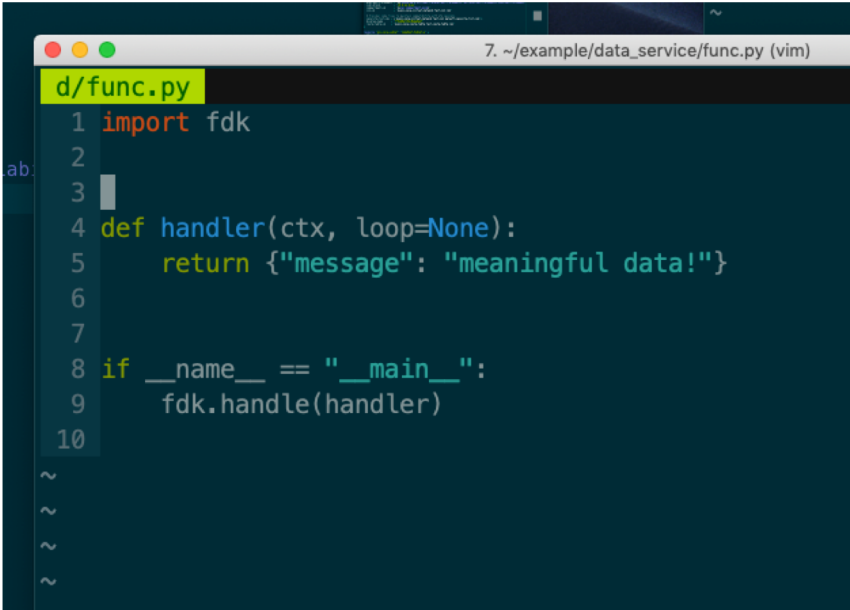
Functions and Serverless

- Serverless doesn't strictly mandate the use of functions
- Would-be microservices are implemented as functions
- Functions are mapped to events
 - API calls
 - Object store actions
 - Messages arriving in a queue



Function Example

- Different projects and products differ in use and workflow
- This example function can be deployed in Oracle Functions
- Just the code, configured against any number of event triggers
- As with microservices, applications are composed of many functions



```
7. ~/example/data_service/func.py (vim)
d/func.py
1 import fdk
2
3
4 def handler(ctx, loop=None):
5     return {"message": "meaningful data!"}
6
7
8 if __name__ == "__main__":
9     fdk.handle(handler)
10
```

Fn Project

- Open source serverless compute platform
- Cloud-agnostic, or on premises
- Container-based functions platform
- Native CloudEvents support
- Active project w/ 3500+ commits across 70+ contributors
- Enterprise focus: Secure, Scalable, Observable



Oracle Functions

Oracle Functions

Functions-as-a-Service

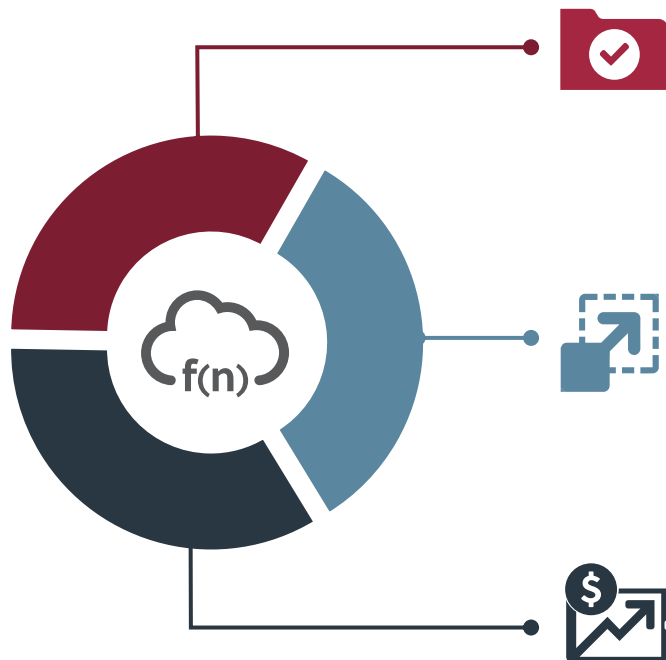
Oracle Cloud Integrated

Container Native

Open Source Engine

Multi-tenant

Secure



Pay Per Use

Pay for execution, not for idle time

Autonomous

Platform auto-scales functions
No servers to provision,
manage

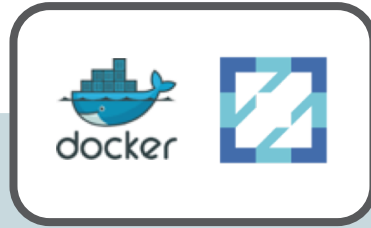
No Lock-in

Built on open-source Fn
Project and Docker

Key Features



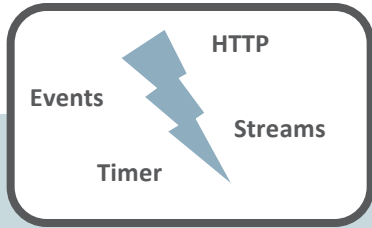
Open Source Engine



Container Native



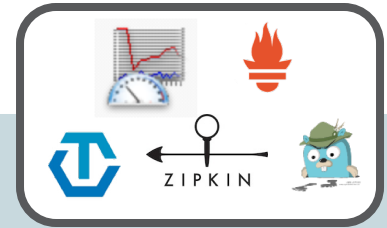
Function Dev Kits



Oracle Cloud Triggers

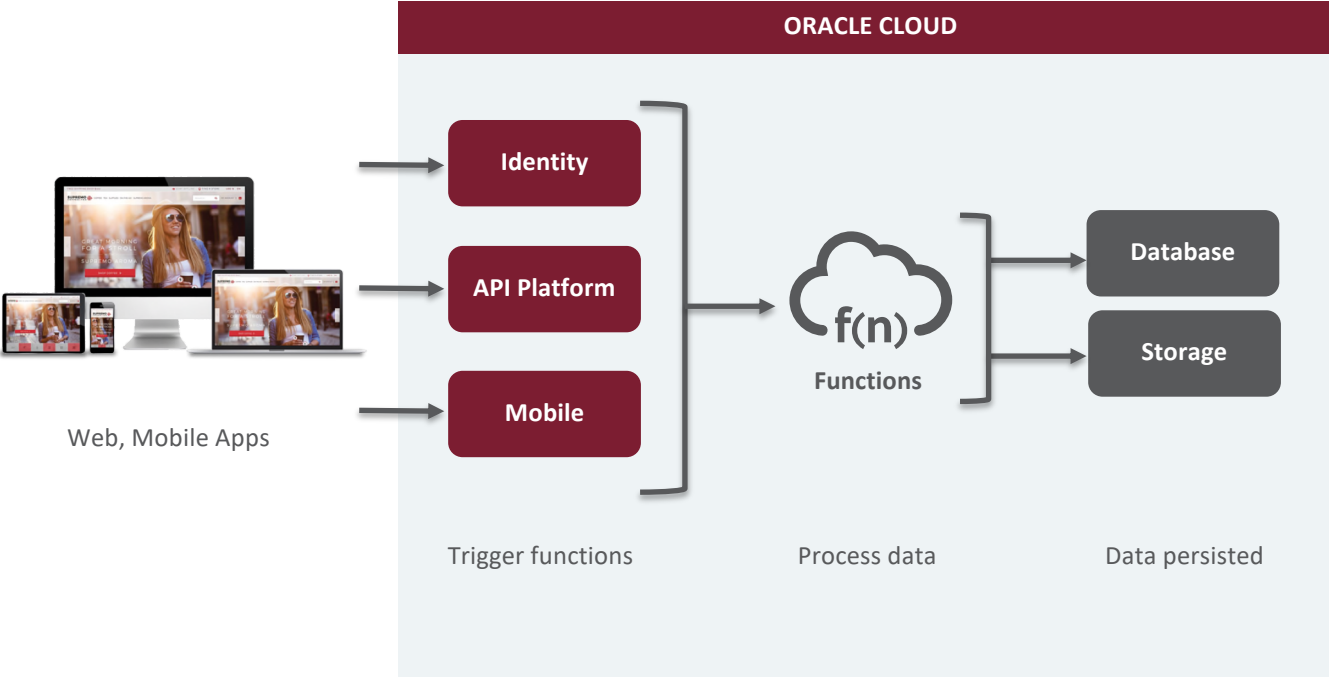


Fine-grained Billing

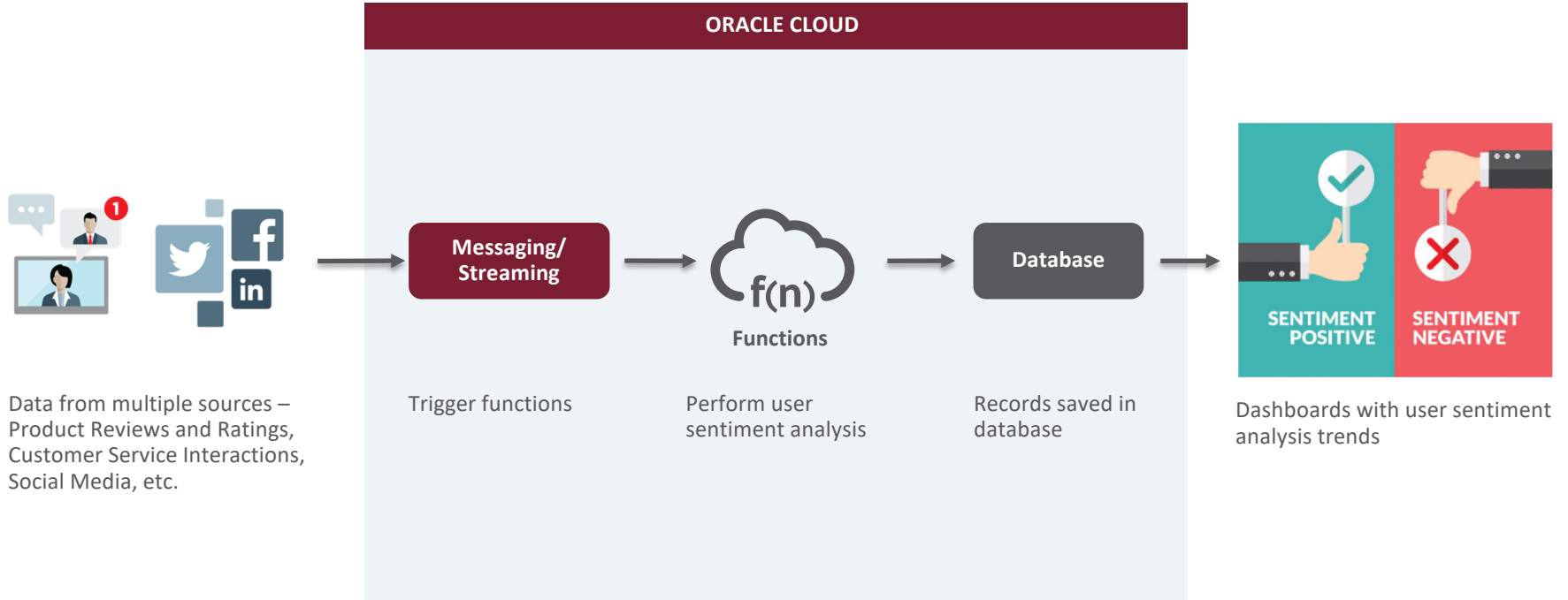


Advanced Diagnostics

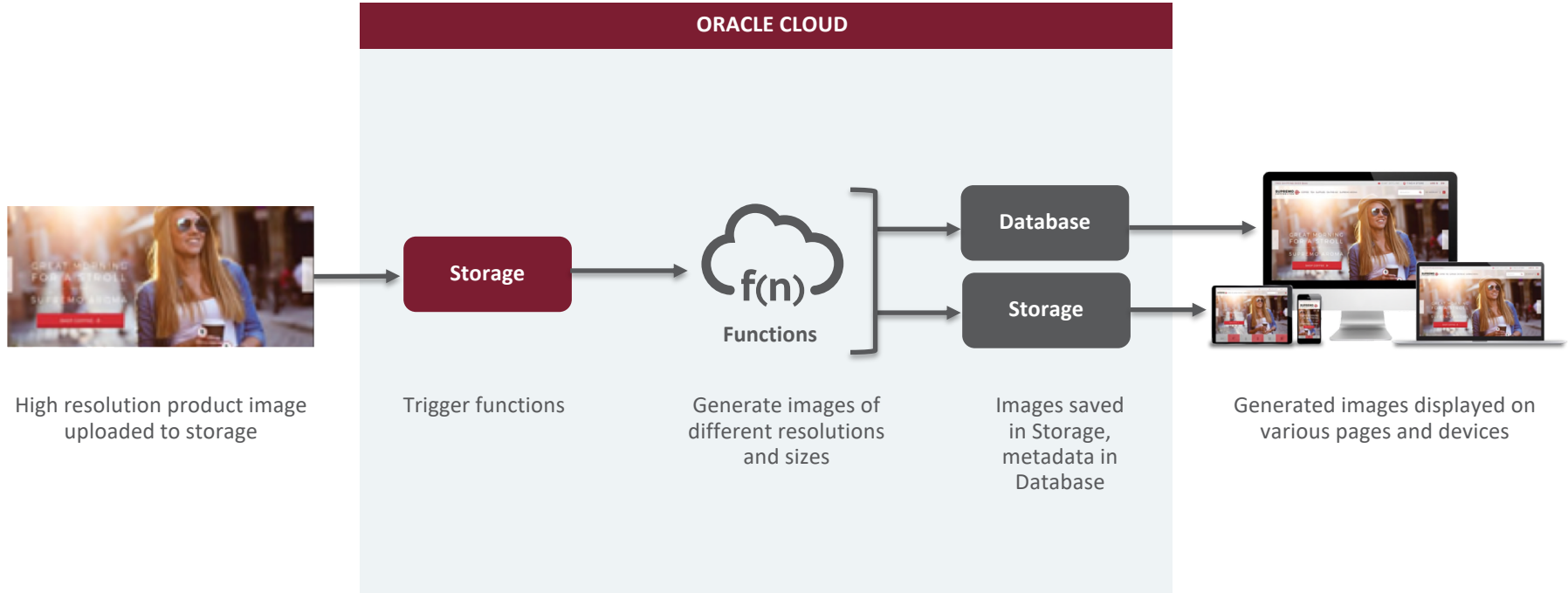
Use Case: Web and Mobile Backends



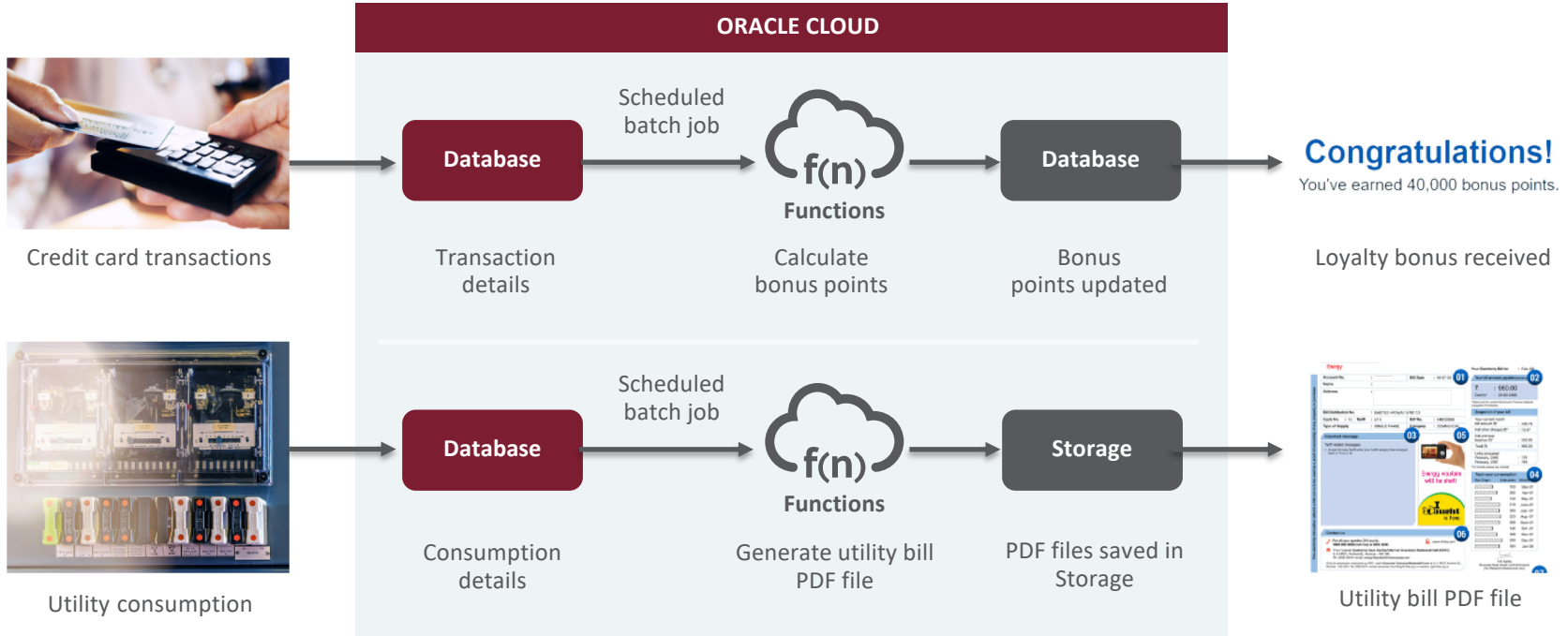
Use Case: Real-Time Stream Processing



Use Case: Real-Time File Processing

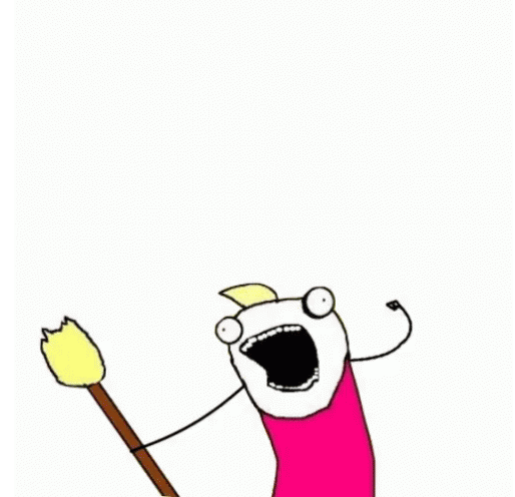


Use Case: Batch Processing



Serverless: Cloud Native Path 2, Brave and Rewarding

- Serverless allows teams to focus on code and business logic, infrastructure becomes invisible
- Leverages all we have learned over the last decade, but it's automated and abstracted
- Can significantly reduce costs, given proper planning and implementation
- Fastest path to Cloud Native from the Old World



Microservices or Serverless?

- Maintaining your own fleet of containers provides a great deal of flexibility and offers the best opportunities for observability and monitoring
- Though the learning curve can be steep, there are solutions for every common production need, and a huge and helpful community
- Managed cloud services like OKE on OCI make Kubernetes easy to deploy and puts most of the complexity on us, letting you focus on building your systems and deployments

Microservices or Serverless?

- Serverless is likely a faster and simpler path to cloud native, especially for some applications, with fully-managed invisible infrastructure
- Not paying for idle time almost guarantees a reduction in OpEx, often a significant reduction
- Serverless softly enforces many cloud native best practices, allowing teams to get productive quickly with far less learning curve

So, really... Microservices or Serverless?

- These are not mutually exclusive approaches and in fact can compliment each other well
- Applications and services which require specific infrastructure components are likely to remain in the managed container space for some time
- Applications which are primarily backend implementations behind an API should strongly consider going straight to serverless
- Moving forward, greenfield projects should always compare the two



ORACLE[®]
Cloud Native Labs

Thanks!

cloud.oracle.com/tryit
cloudnative.oracle.com