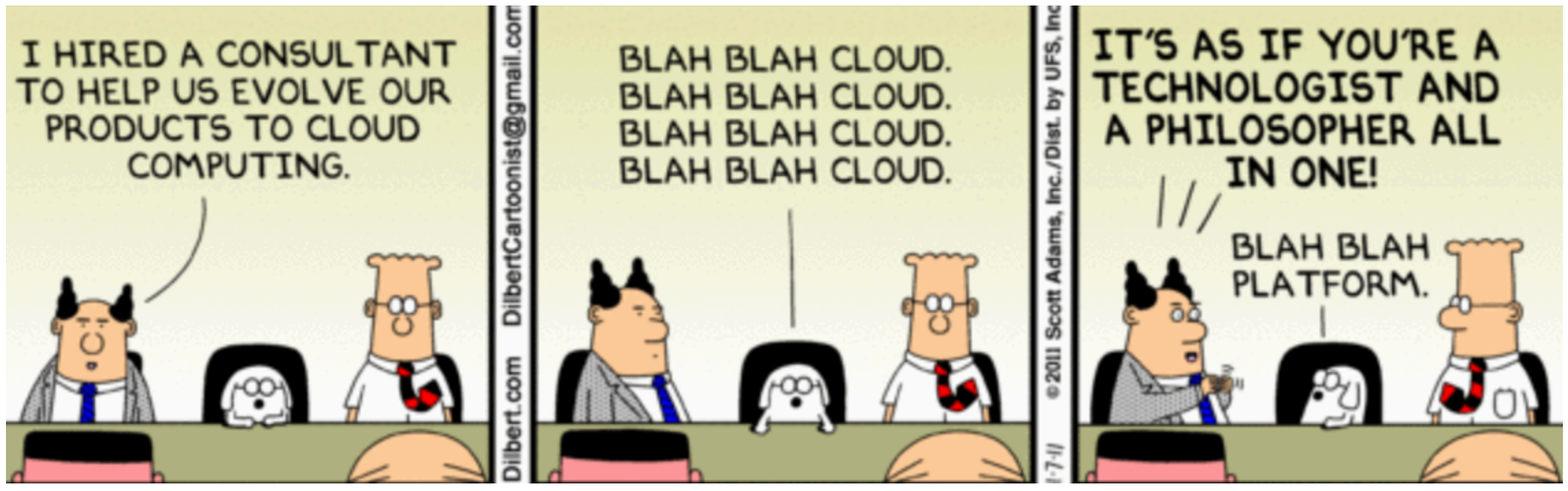
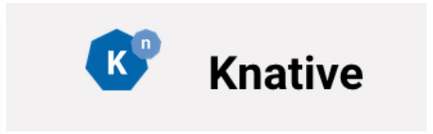
DHBW Informatik Semester 4
Wahlpflicht
Cloud Computing
Chapter 3 PaaS towards Serverless
Or what comes next on top of Kubernetes

Juergen Schneider



Agenda



- What is next on top of Kubernetes
- The Principles of Cloud Foundry and Serverless.

Why should I care about containers ?

- All I need is a means to give somebody my code and have it run in the cloud…. (a Developer view)
- Containers (PaaS but with the need of infrastructure skill)
  - More control of the deployment, the resource usage (and limits) the network traffic and security
  - But complicated, needs skill in infrastructure
- PaaS (the '*real*' PaaS)
  - 'Explain the Code' and a **bit of infrastructure** and **accept** the rest to be done by the cloud (platform layer) and admin
    - Service Access is HTTPS or another IP Layer 4 Protocol with a generated subdomain.. (TLS Certificates available or configurable)
    - No control of what is virtualized or not
    - Data (only) via Data (Database) Services (typically no local disk recovery)
    - Programming model driven by the 12 factor directives

What does a Developer and his/her admin really need

Stateless cloud native App

Developer:

- Name of the App
- # of Instances and Scaling Rules
- Mem. CPU in T-shirt sizes
- External and or Internal Network
- Environment Variable
- Start Command
- Services needed and used
- Code Repository (here is my code)
- Concurrent Updates ?

Admin:

- Own Domain or system generated Sub-Domain
  - TLS Certificate
  - DNS
- Image Registry URL and Secret
- Location
- Secret Tokens (as Environment Variable)
- Service Binding Setup
- Own Dockerfile or pre canned buildpacks
- Sidecar Setup
- Restart Policy

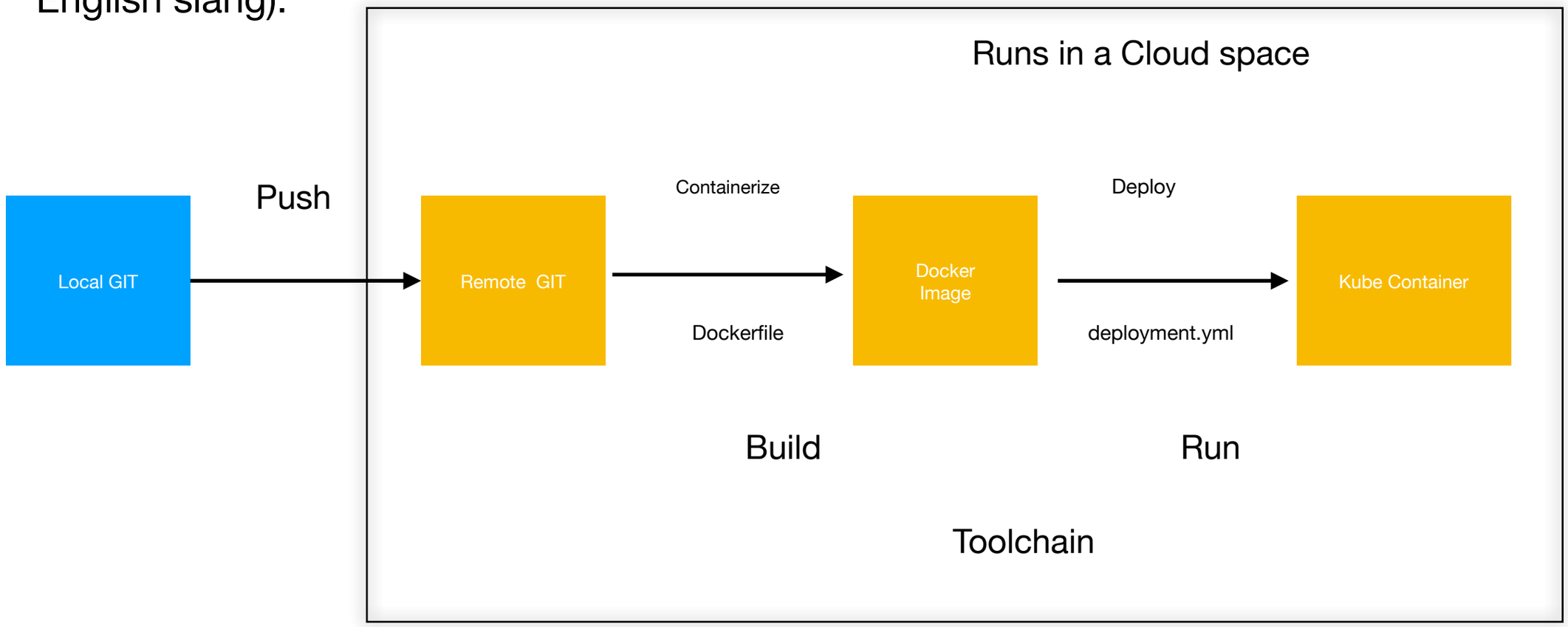
Somebody else

- Multizone Cluster
- Additional Volumes
- Backups
- Special HW (for worker node)

Enhance Kubernetes to allow a developer Usage (Concept Example)

Add development tool chain integration on top of Kubernetes

- Let a admin person deal with all the YAMLs,
  - the proper network setup, the container, the images, the side-car etc.
- Let the developer just use a Code Repository to push code into the cloud.
  - The cloud configuration is in files of the **remote** repository but not in my code (12 factor app)
  - Torvalds sarcastically joked about the name *git* (which means "unpleasant person" in British English slang):



PaaS PaaS++(FaaS)

Dev / Ops (User) provides it

Somebody else manage it

Developer

Admin /Operator

Kubernetes + Toolchain

Code

Build/Run

Dockerfile

Deployment.yml

Cloud Foundry

Code

Manifest.yml

Build/Run

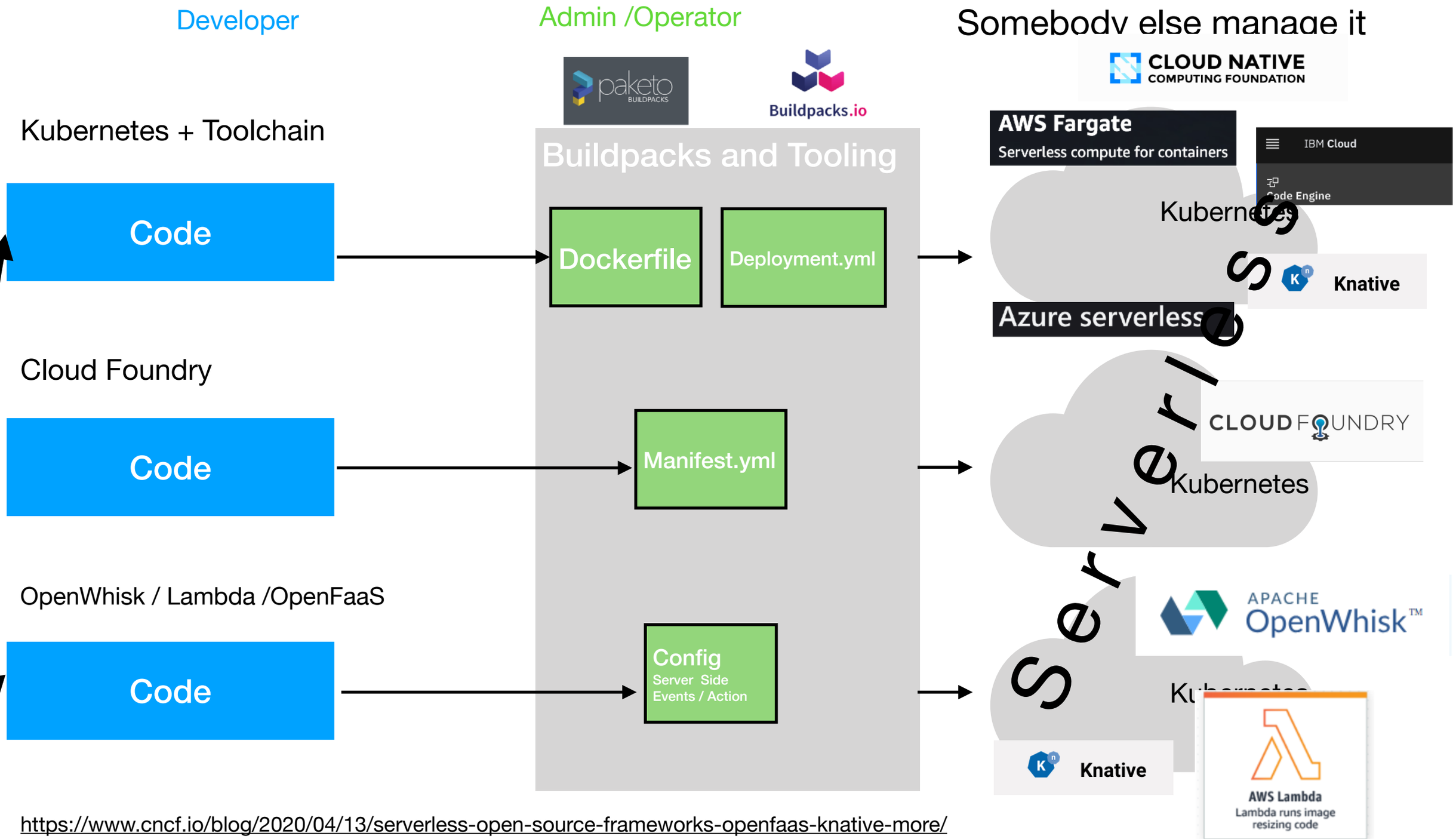
OpenWhisk / Lambda //OpenFaaS

Code

Config
Server Side Events / Action

Build/Run/destroy on demand/event

The current Market Place Battle (biased)



https://www.cncf.io/blog/2020/04/13/serverless-open-source-frameworks-openfaas-knative-more/

https://knative.dev/docs/

https://paketo.io/

Serverless ? - Cloud Provider View

https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-serverless-computing/

Serverless application patterns

Developers build serverless applications using a variety of application patterns—many of which align with approaches that are already familiar—to meet specific requirements and business needs.

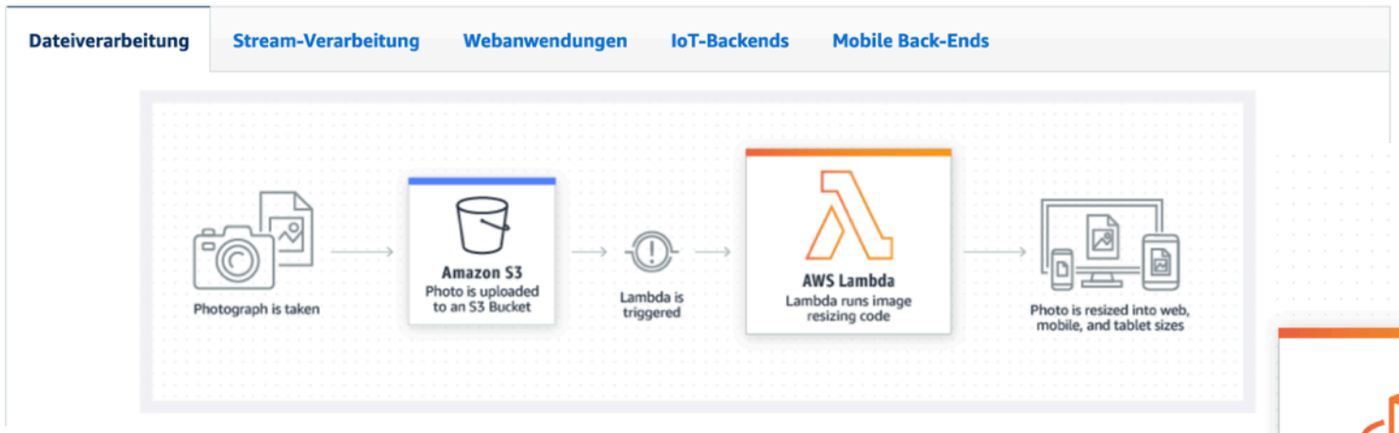
Serverless functions

Serverless functions accelerate development by using an event-driven model, with triggers that automatically execute code to respond to events and bindings to seamlessly integrate additional services. A pay-per-execution model with sub-second billing charges only for the time and resources it takes to execute the code.

Serverless Kubernetes

Developers bring their own containers to fully managed, Kubernetes-orchestrated clusters that can automatically scale up and down with sudden changes in traffic on spiky workloads.

https://aws.amazon.com/de/lambda/

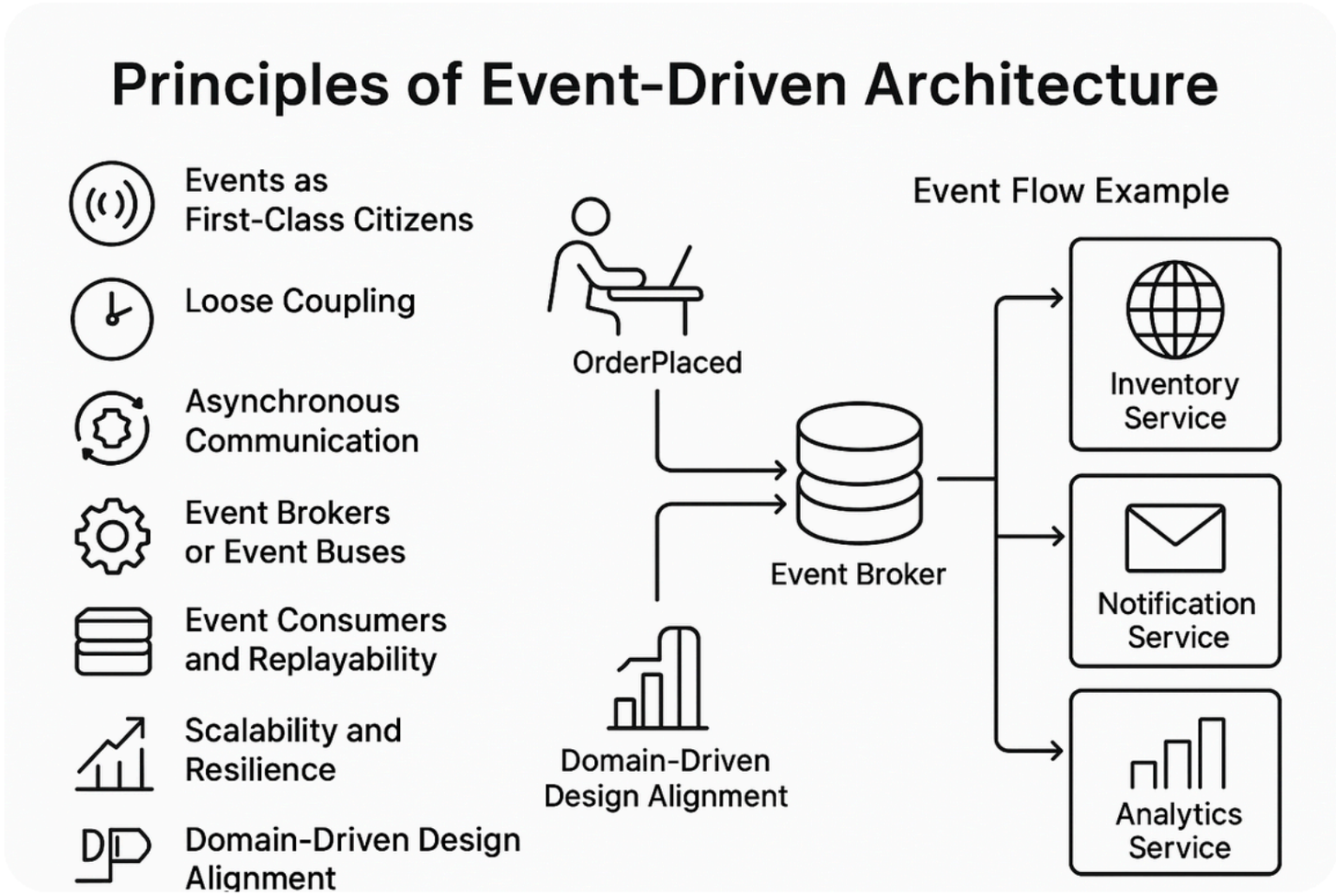


https://aws.amazon.com/de/fargate/?c=ser&sec=srv

I can also ‘buy’ and ‘pay’ for my own Kubernetes Cluster

Event Driven Architecture

Why not ask ChatGPT, here is the result:



Events as First-Class Citizens

- An **event** is a significant change in system state (e.g., "user placed an order").
- Events are published and consumed as the primary mechanism for communication.

Loose Coupling

- Producers (senders) of events do not know who will consume them.
- Consumers (listeners/subscribers) process events independently.
- This promotes flexibility and scalability.

Asynchronous Communication

- Events are typically processed asynchronously.
- Producers do not wait for consumers to finish processing.
- Improves system responsiveness and decouples processing speed.

Event Brokers or Event Buses

- Middleware like Kafka, RabbitMQ, HTTP, or AWS EventBridge routes events from producers to consumers.
- Helps manage scalability, filtering, and delivery guarantees.

Event Persistence and Replayability (optional)

- Events can be stored for auditing, debugging, or replaying system state.
- Event Sourcing is a related concept where application state is derived from a sequence of events.

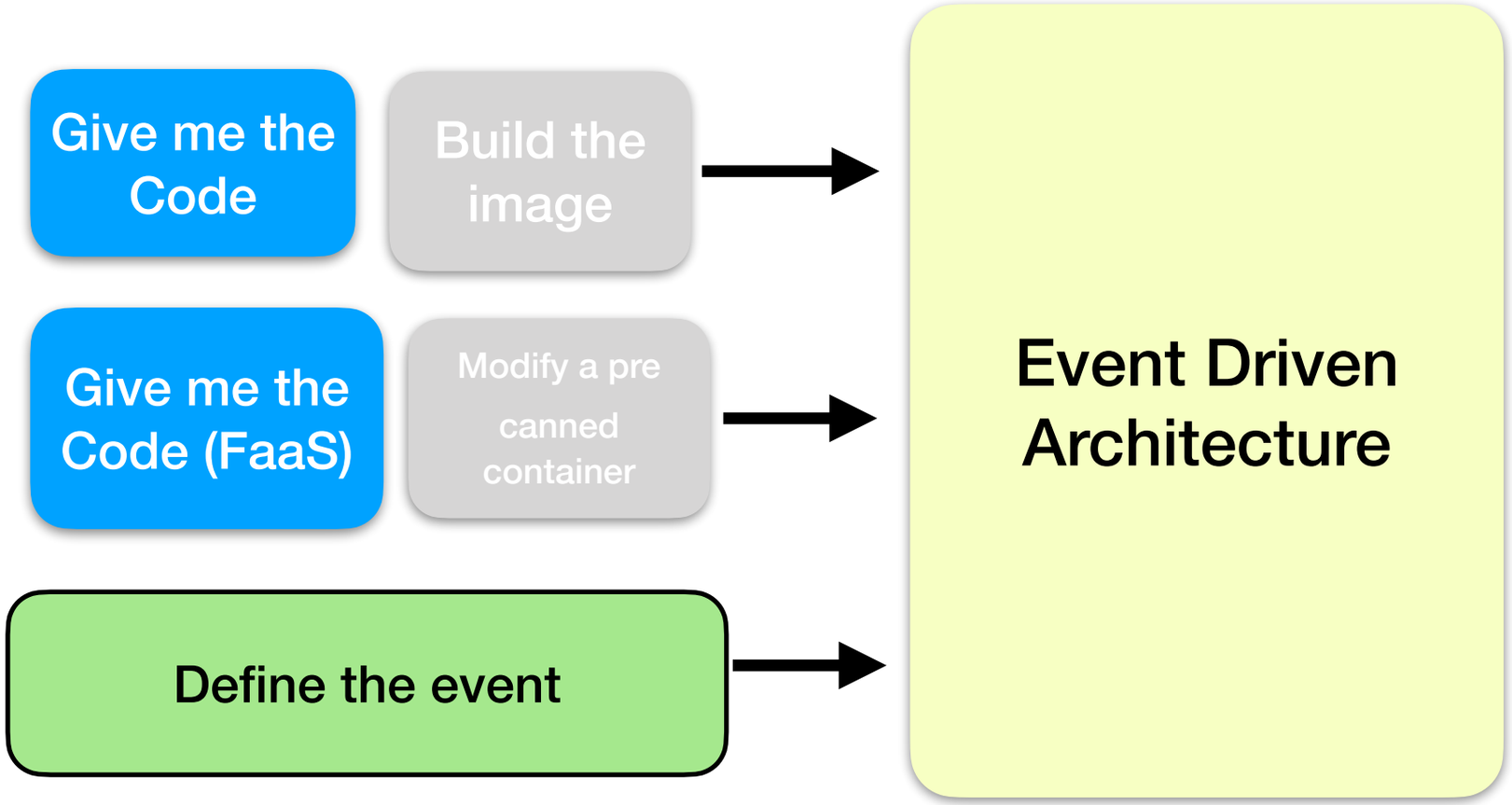
Scalability and Resilience

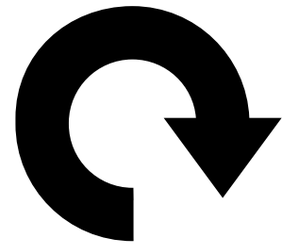
- **The architecture naturally supports horizontal scaling.**
- Components can fail independently, improving fault tolerance.

Domain-Driven Design Alignment

- Events often map closely to business domain events, aligning the system behaviour with business operations.

Event Driven Architecture —> Serverless





Scale up and down container

Serverless

Open Source View

CLOUD NATIVE COMPUTING FOUNDATION

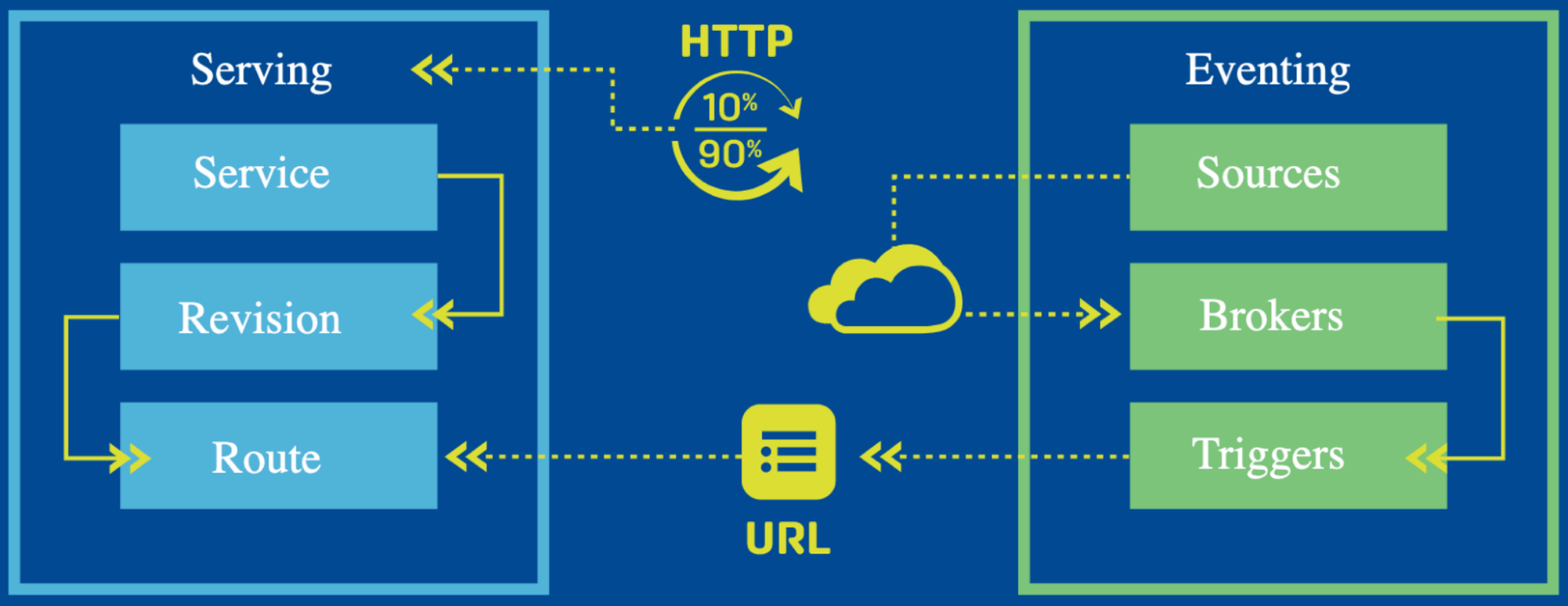
Knative

https://www.cncf.io/

https://knative.dev/docs/

Knative Components

Knative has two main components that empower teams working with Kubernetes. Serving and Eventing work together to automate and manage tasks and applications.



Run serverless containers in Kubernetes with ease. Knative takes care of the details of networking, autoscaling (even to zero), and revision tracking. Teams can focus on core logic using any programming language.

Universal subscription, delivery and management of events. Build modern apps by attaching compute to a data stream with declarative event connectivity and developer friendly object models.

IBM Code Engine - a Knative Implementation

Workload : see next charts

- Applications, Jobs, Functions

Event Sources

- Cron, IBM Object Storage (COS), Kafka, Webhooks (Github) .. more to come

Event Brokering

- related to the Event Source
    - Kafka -> Topics,
    - COS -> Updates for Bucket entries (create, read, update, delete)
    - HTTP (Url)
- Trigger event data follows the cloud events specification: https://cloudevents.io/

Revisions:

- Revisions represents snapshot-in-time of application code and configuration. Revisions enable progressive rollout and rollback of application changes by changing the HTTP routing between the Revision instances.

Applications - Jobs - Functions

Triggers

Cron

HTTP

Messages

DB Update

. . .

Web Hook

Integration with Git Repository

For Image Build

Application

Long running
Port listening
Multiple request/threads

Scaling based # request (min/max)

Jobs

Very long running
No Port listening

Scaling based specific workload

Functions

short running
No Port listening
Single request/thread

Scaling based on request (0 to current)

Source

Image

Bring your image

- build by Dockerfile or buildpacks

Pre-canned Containe

(Language/Capacity)

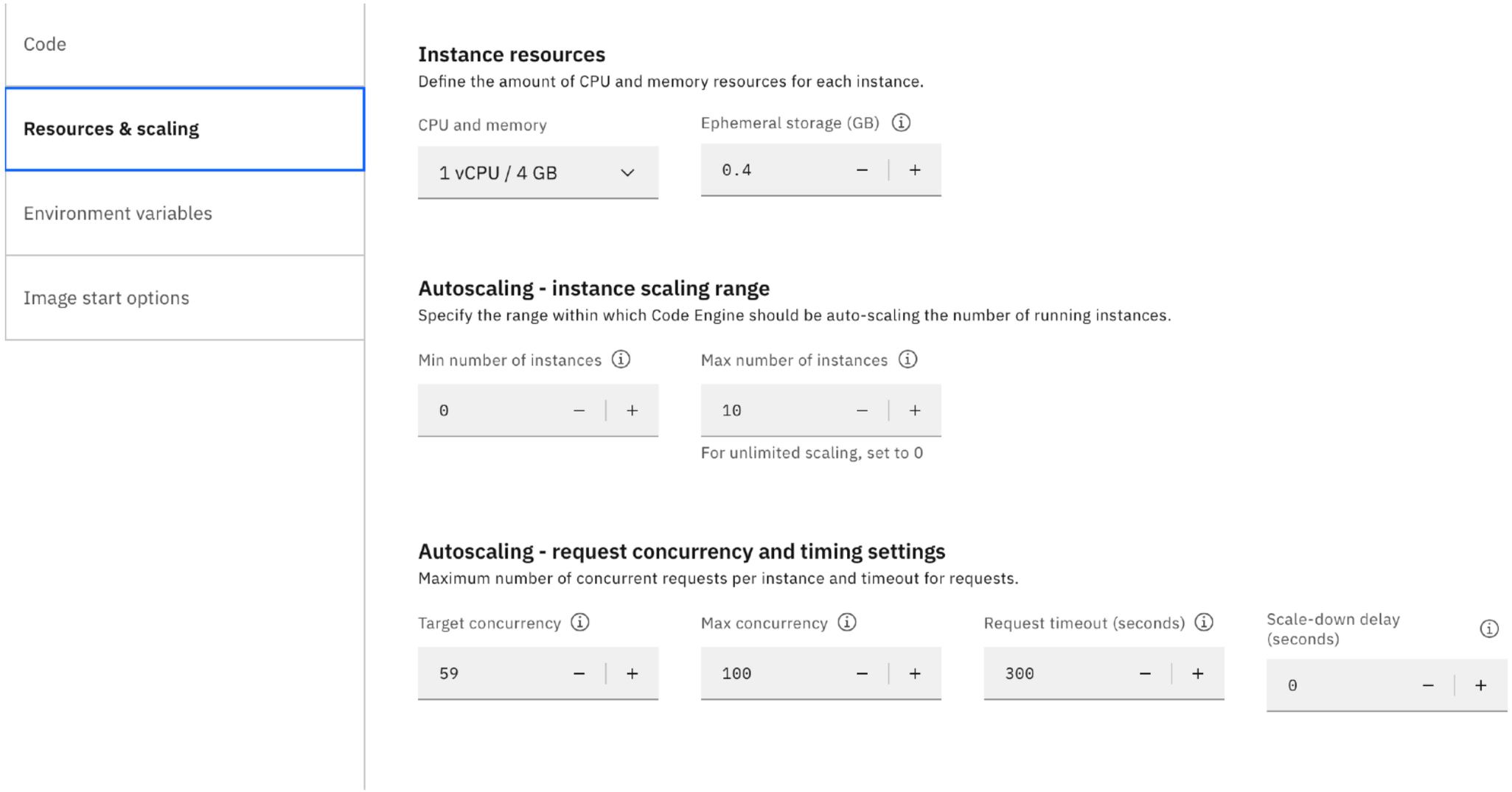
Code

Bundles

```
main.js

function main(args) {
  const body = {
    args,
    env: process.env,
  };

  console.log(`Return body: ${JSON.stringify(body, null, 2)}`);

  return {
    statusCode: 200,
    headers: {
      'Content-Type': 'application/json',
    },
    body,
  };
}

module.exports.main = main;
```

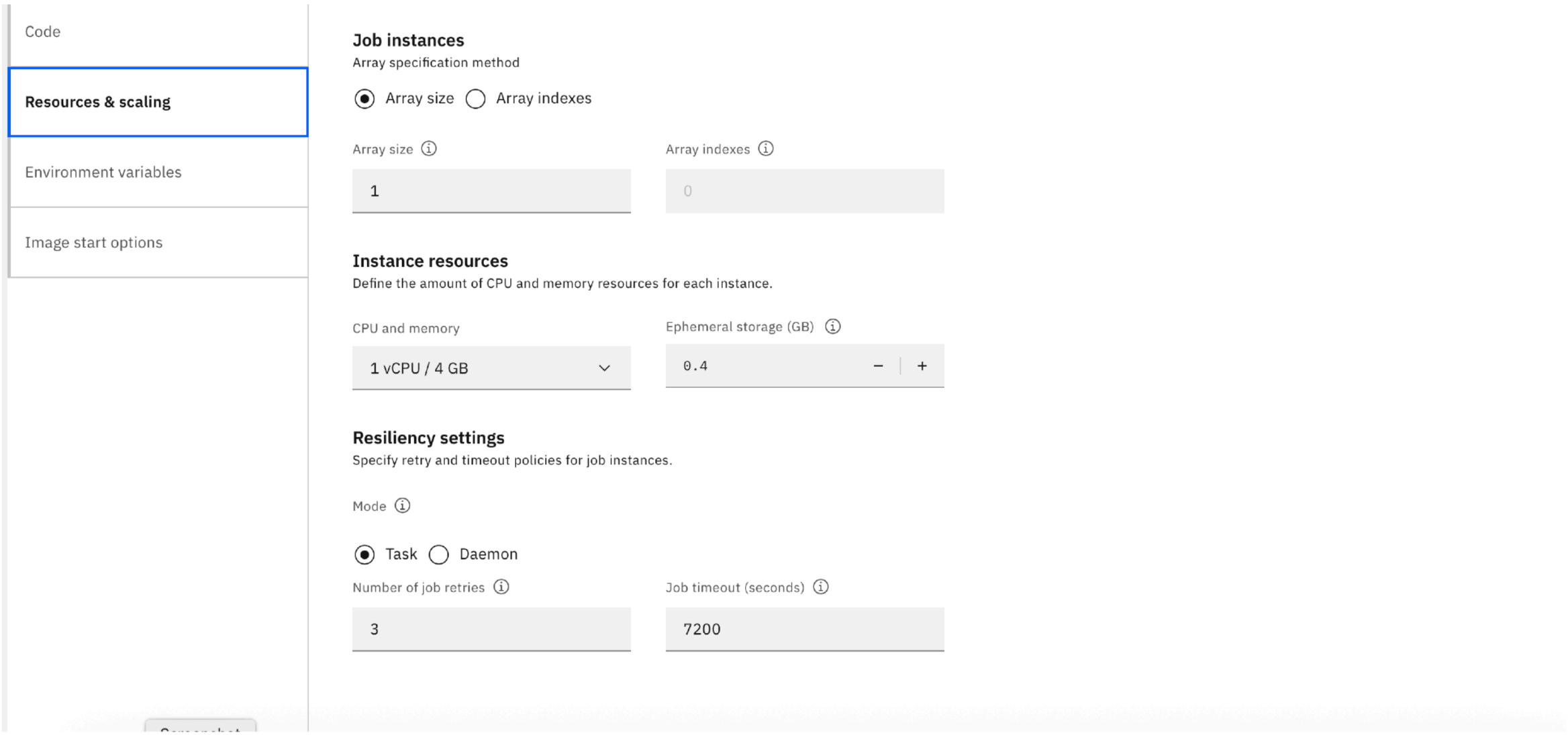
Bring your code - inline or Github

Capacity and Scaling Profiles for Application



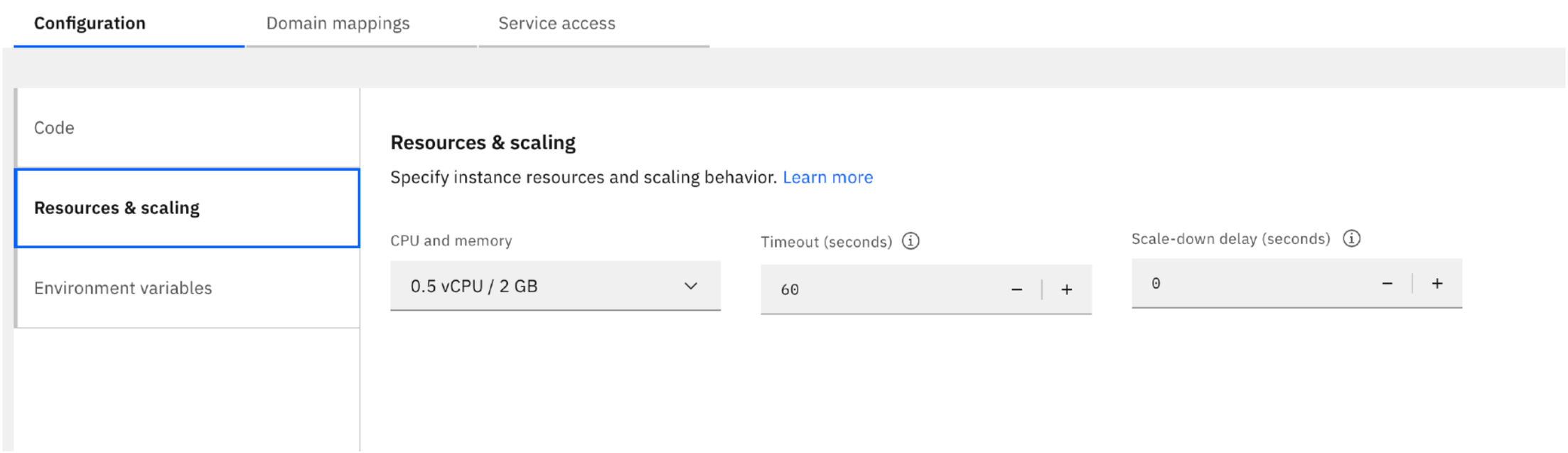
- **Scaling Range** : min/max number of instances
- **Target concurrency** : average concurrent requests to start scale up to avoid max concurrency
- **Max concurrency** : Hard limit to start to scale up
- **Request timeout** : a time in which the app should respond, before the requestors may fail. As consequence the request will be terminated (HTTP Status 503). And it may also be an indication that something is wrong
- **Scale down delay**: Number of seconds to wait (delay) before scale down (as the number of concurrent requests proposes a scale down)

Capacity and Scaling Profiles for Jobs



- **Job Instances :** you can split the Batch/Job workload in more than one instances, if that workload allows it and each instance could have a unique index as input (Job Arrays). Each instance will run once.
- **Task or Daemon :**
  - In Daemon mode: the job runs without a maximum execution time, and failed indexes are restarted indefinitely.
  - In Task mode**: Number of retries** a single Job (Task) are restarted before considered as failed and **Job Timeout** reflects the time a single Job can run (maximum execution time)

Capacity and Scaling Profiles for Functions



- For each incoming request a new container with the function code will run
- **Timeout :** how long is a single function allowed to run.
- **Scale down delay:** how long to wait until I give away an exited container, as maybe new requests come in soon

Demos :

1. Run a Job which sends out an email, of there is an update on COS DB

2. Trigger a Function which inverse the text given

3. Trigger a Function the speak the text given (Cloud Service)

4. Run our (almost) famous tinyApp as Serverless App

5. Demo a GIT (Webhook) to build an new image for our App

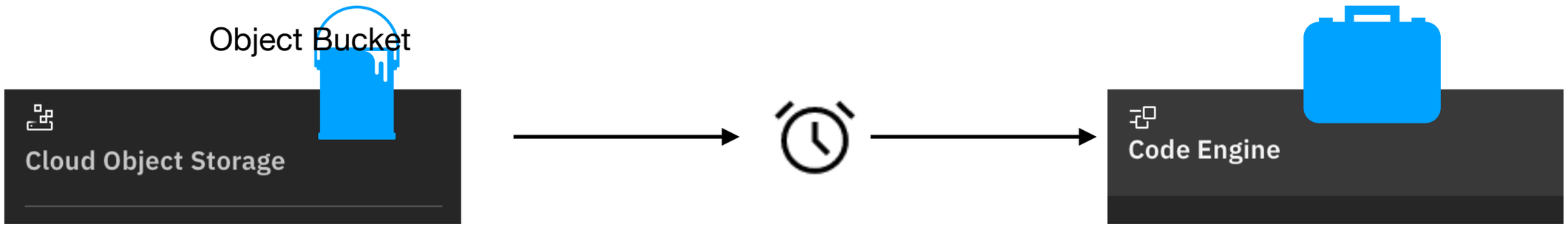
Example : Event Driven Invocation of Jobs in Code Engine

Demo : an Update of an COS bucket generates a email

Node.js Server in InternalStuff/emailserver

Job = notifyme .. data given via cloud events /

Environment variable CE_DATA




```
ic ce project select -n DHBW-Code-Engine-Project
ic ce job update --name notifyme --build-source . --strategy buildpacks
```

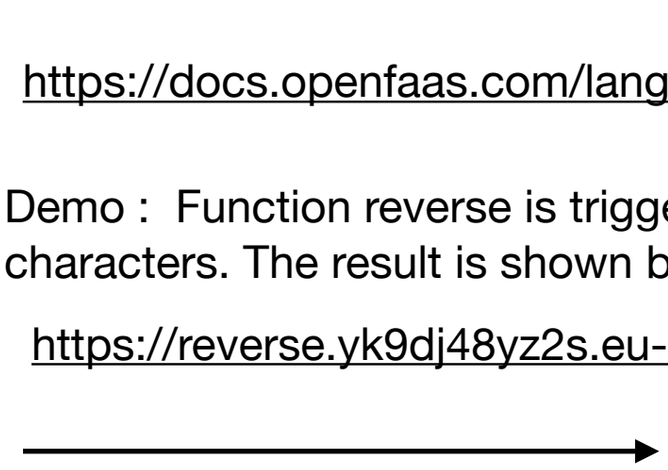
Demo commands to be used in InternalStuff/CodeEngine/emailserver commands.txt

A basic Code Engine Function Example based on OpenFaaS

https://docs.openfaas.com/languages/node/

Demo : Function reverse is triggered by a HTTP Get and takes the ?text=data query string and reverse the characters. The result is shown back to the caller (browser)

https://reverse.yk9dj48yz2s.eu-de.codeengine.appdomain.cloud?text=Geht es nun wirklich besser



Node.js 18

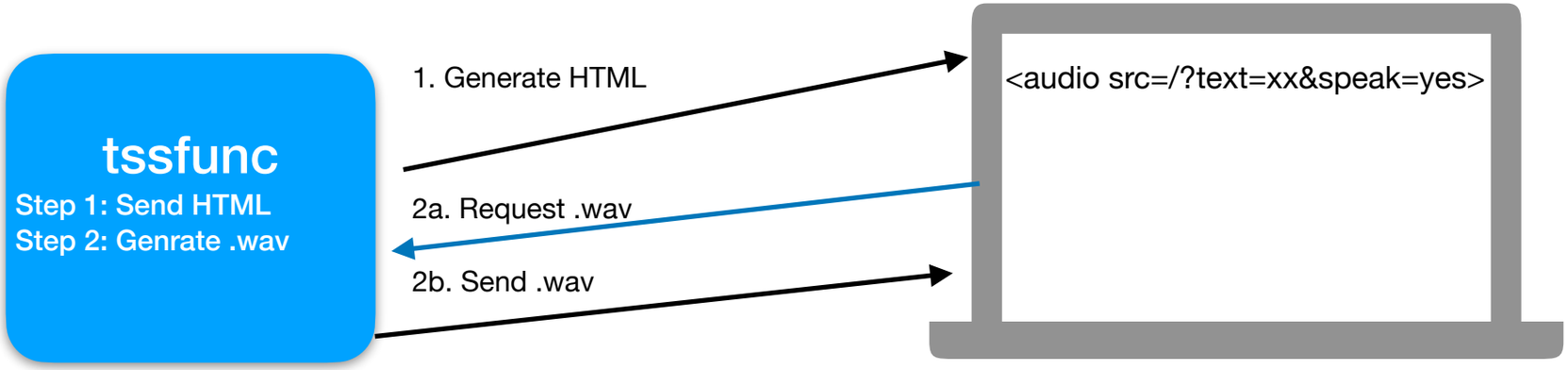
main.js

```
function main(args) {
  const body = {
    result : (args.text == undefined) ? "Bitte mit dem richtigen Parameter" : args.text.split("").reverse().join(""),
 };

  console.log(`Return body: ${JSON.stringify(body, null, 2)}`);

  return {
    statusCode: 200,
    headers: {
      'Content-Type': 'application/json',
    },
    body,
  };
}

module.exports.main = main;
```

A more sophisticated Code Engine Function based on OpenFaaS

Demo :

- Function sends a HTML document including an audio tag with an URL which again call this function to synthesise the audio using a IBM Cloud Service.
- As this function is asynchron in nature the function a a asynchronous function
- As this function is using additional node.js packages, the function must be rebuild generating a small mage (with a additional modules)
  - ic ce fn create --name ttsfunc --runtime nodejs-20 --build-source .
- The credentials to a Cloud Services are securely provides as Environment key/values (Kubernetes Secret File) and never externalised

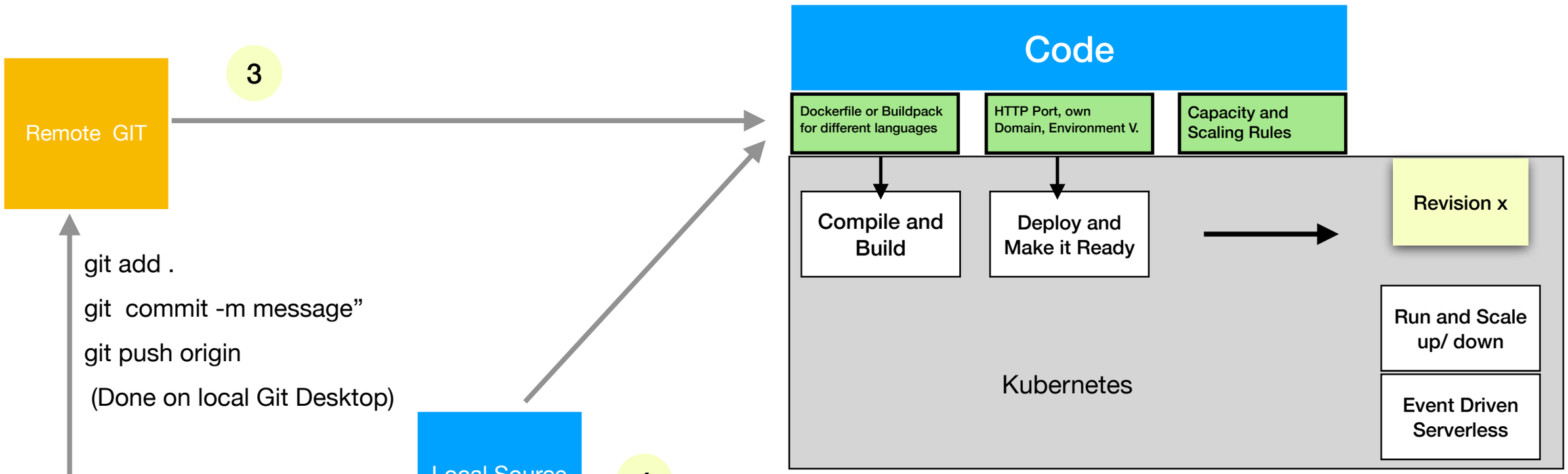


Show Code Engine GUI with Service Binding, Environment Variables

https://ttsfunc.yk9dj48yz2s.eu-de.codeengine.appdomain.cloud?text=Geht es nun wirklich besser

—> in Chrome oder Firefox.. Safari audio Format führt zu Fehlern

Building an App either using a local build or as part of a Git Push




```
<Cloud2025/internalStuff/tinyNodejsApp>
ic ce project select -n DHBW-Code-Engine-Project
ic ce app update --name myappdhbw --image docker.io/juergenschneider/ultimativeapp:dhbw --port 80
ic ce app update --name myapplocal  --build-source . strategy dockerfile -port 80
```

https://paketo.io/

3 <any directory>

ic ce app update --name myappgit --src git@github.com:JuergenSchneiderDHBW/tinyNodejsApp.git --repo repro-access-1 --strategy buildpacks

Summary on Workload Type

https://cloud.ibm.com/docs/codeengine?topic=codeengine-cefunctions

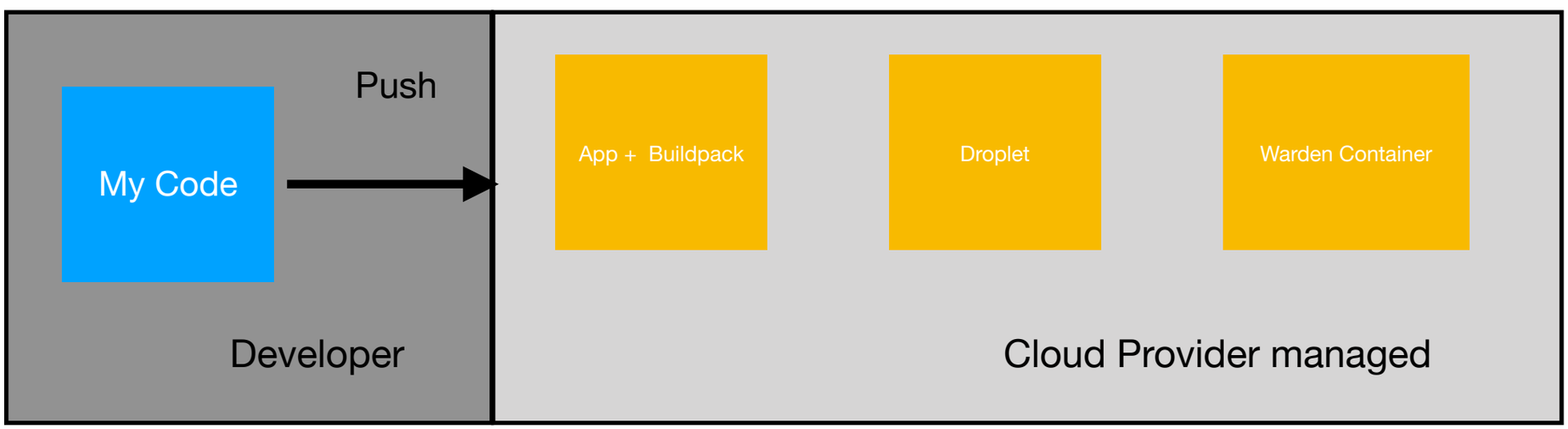
How do functions compare to apps and jobs?

Characteristic	App	Job	Function
Execution time (duration)	Long-running (10 minutes per request)	Long-running (up to 24 hours)	Short-running (2 minutes or less)
Startup latency	Medium	Scheduled start	Low
Termination	Run-continuously	Run-to-completion	Run-to-completion
Invocation	On request or permanently running	Scheduled	On request, instant
Programming Model	Container-based build and execution	Container-based build and execution	Language-specific source code files and dependency metadata
Parallelism	Parallel execution, flexible	Low to medium parallel execution	High parallel execution
Scale-out	Based on number of requests	Based on job workload definition	Based on events or direct invocations
Optimized for	Long running, highly complex workload and on-demand scale-out	Scheduled or planned workloads with high resource demands	Startup time and rapid scale-out

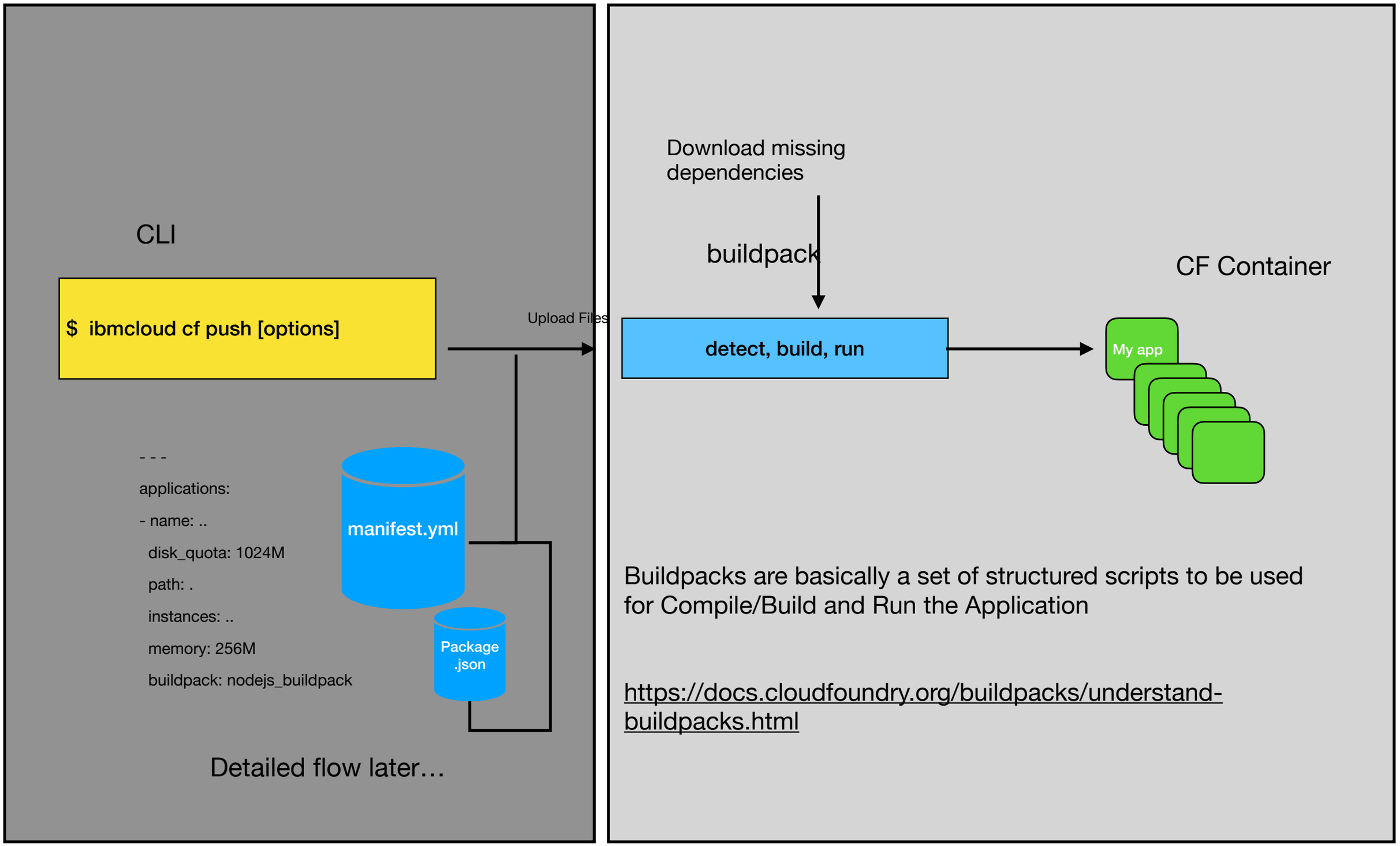
Collapse

Cloud Foundry

- History
  - Started in 2009 with a small team in VMware
  - First announced April 2011
  - April 2012 Usage of BOSH https://www.bosh.io/docs/ as a development toolchain for large distributed systems to deploy Cloud Foundry PaaS
  - April 2013 Pivotal was founded by EMC and VMware to become the owner of Cloud Foundry
  - April 2014 Open Source governed.. with a broad coverage of important players (Platinum Level Cisco, Dell/EMC, IBM, SAP, Pivotal, Suse, VMware, Gold Level Alibaba, Google, Swisscom, VW, Ford, Microsoft)
  - https://www.cloudfoundry.org/
  - **Cloud Foundry lost the game against Kubernetes and their huge ecosystem to provide exactly this capabilities**
    - https://krishnan.medium.com/lessons-from-the-demise-of-cloudfoundry-ed4f77a08a73
- Here is the code, please build an executable and run it
  - CLI c**f push** (manifest.yml).
    - Build and Compile
    - **App + Buildpack** = Droplet (a sort of a image)
- Run
  - **Droplet + CF Runtime** = App running in a container



Cloud Foundry Basic Deploy Flow



Cloud Foundry Buildpack

- provide the runtime support for apps (Result of a build back is a droplet (a ‘image’ ready to get deployed)
- examine the apps to determine what dependencies to download and how to configure the apps to communicate with bound services.
  - In the core it is a set of script flows as described here
    - **bin/detect** checks the code directory to figure out whether the right build pack has been chosen
      - In case of node.js it tests whether a package.json exists
    - **bin/supply** provides all the dependencies
      - In case of node.js it executes the npm install command
    - **bin/finalize** prepares the app for launch
      - In case of node.js it basically setup some environment variable and the final directory
    - **bin/release** provides feedback metadata to Cloud Foundry indicating how the app should be executed.
      - In case of node.js it tells Cloud Foundry to start node.js with NPM start
- Ready to use build packs are available for
  - Ruby, Java, Go, .NET, PHP, Node.js, Python, Staticfile (a NGINX Web Server for static Web Pages (CSS, HTML, JS)

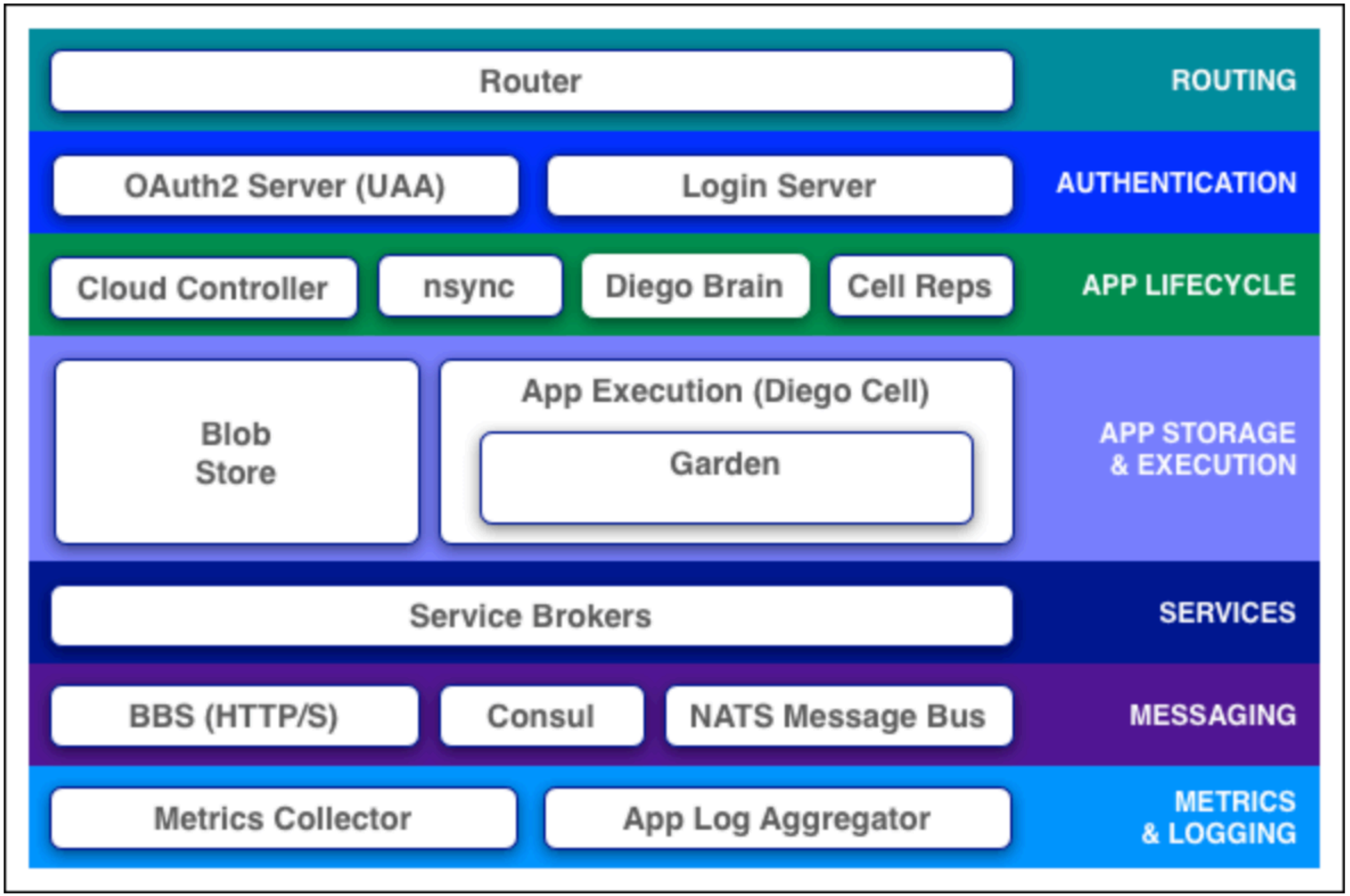
Cloud Foundry Manifest.YML

- Basically all the options you can apply to the CF Push command
  - Application
    - Name of the app (which becomes the subdomain in the dns entry)
    - Memory/CPU, Storage Quota, # of instances
    - Name of the buildpack
    - Start Command (if needed, in node.js this is typical part of the package.json)
    - Environment Variables
    - Directory Path if not just ./
    - Sidecars
    - Services used
    - ….
- CF Push -f manifest

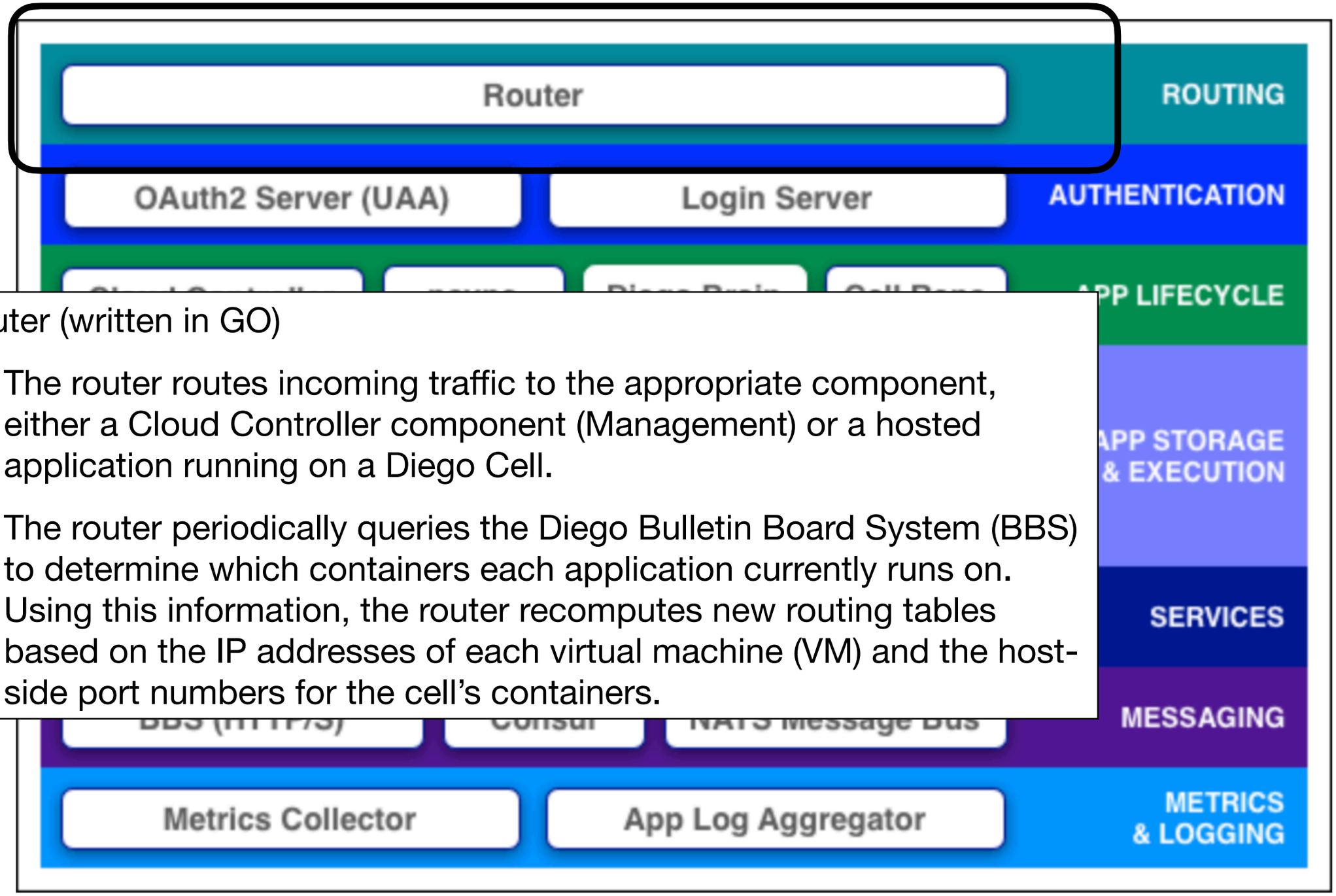
```
---
applications:
- name: my-app
  memory: 512M
  instances: 2
```

CLOUD FOUNDRY

Cloud Foundry Architecture Component Overview



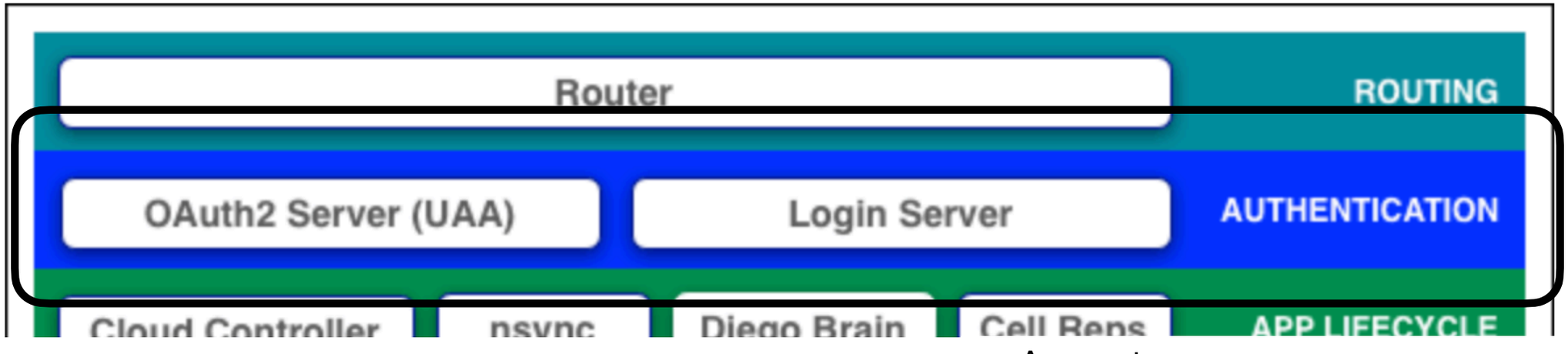
Cloud Foundry Architecture Component Overview

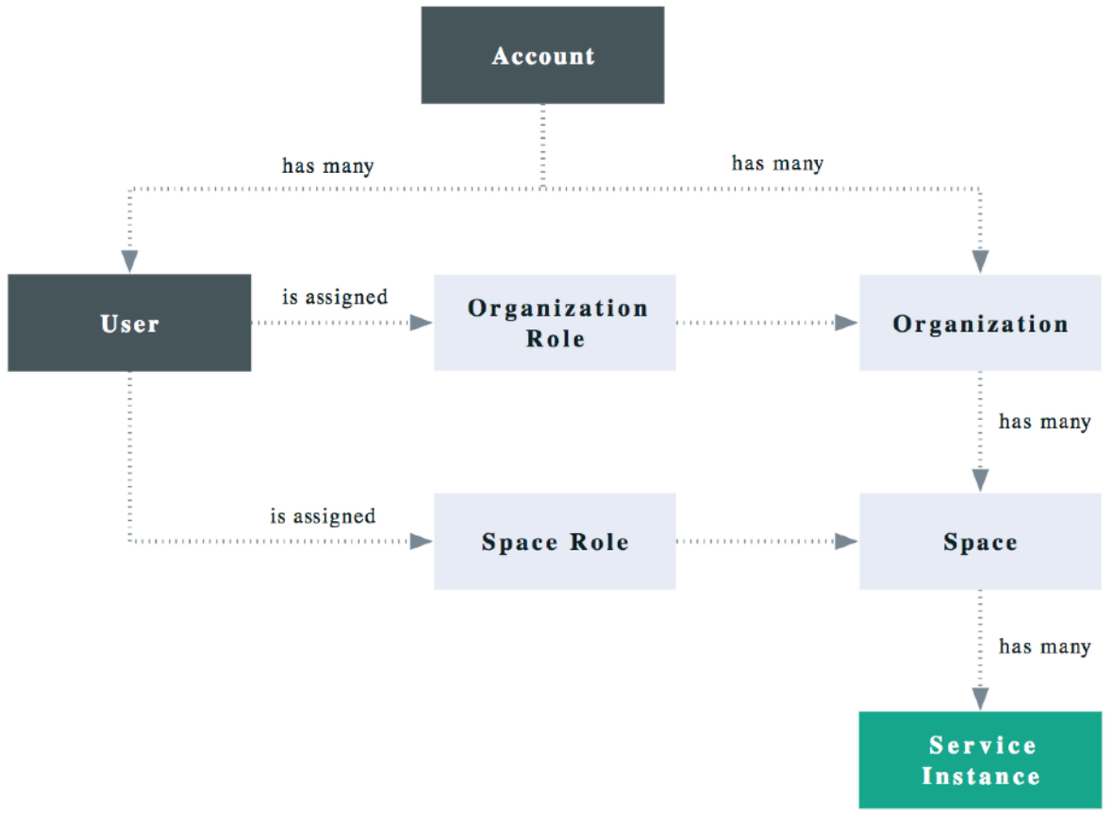


Router (written in GO)

- The router routes incoming traffic to the appropriate component, either a Cloud Controller component (Management) or a hosted application running on a Diego Cell.
- The router periodically queries the Diego Bulletin Board System (BBS) to determine which containers each application currently runs on. Using this information, the router recomputes new routing tables based on the IP addresses of each virtual machine (VM) and the host-side port numbers for the cell's containers.

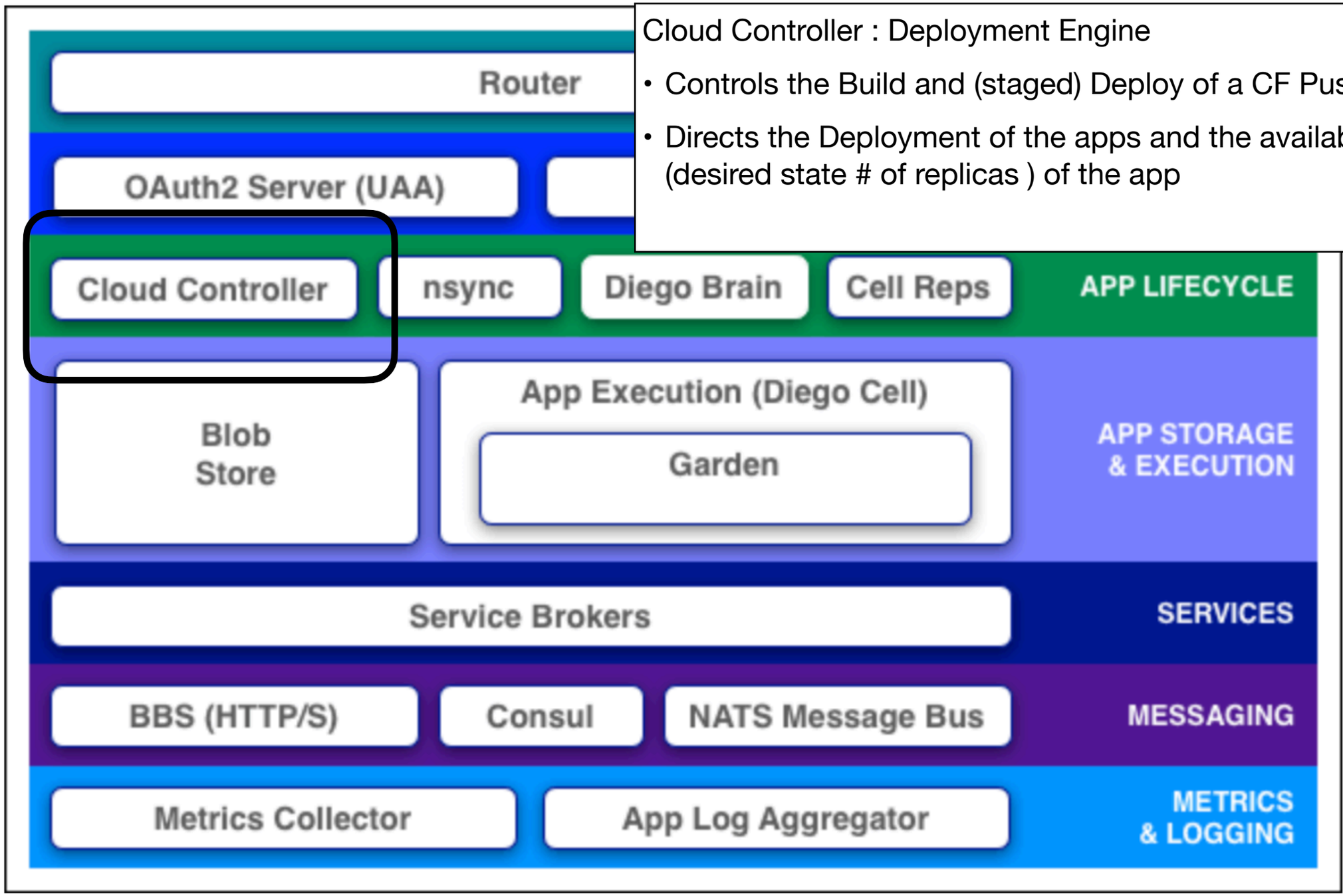
Cloud Foundry Architecture Component Overview





- Account
  - The Cloud Account
- Organizations
  - Within a nAccount one can build many organizations (in our Trial account only 1)
- **Space**
  - A structure in which you can organize different CF projects
- Service Instances
  - A running CF App which belongs to exactly one space in one organization
- User
  - Has a certain role (e.g Manager) with

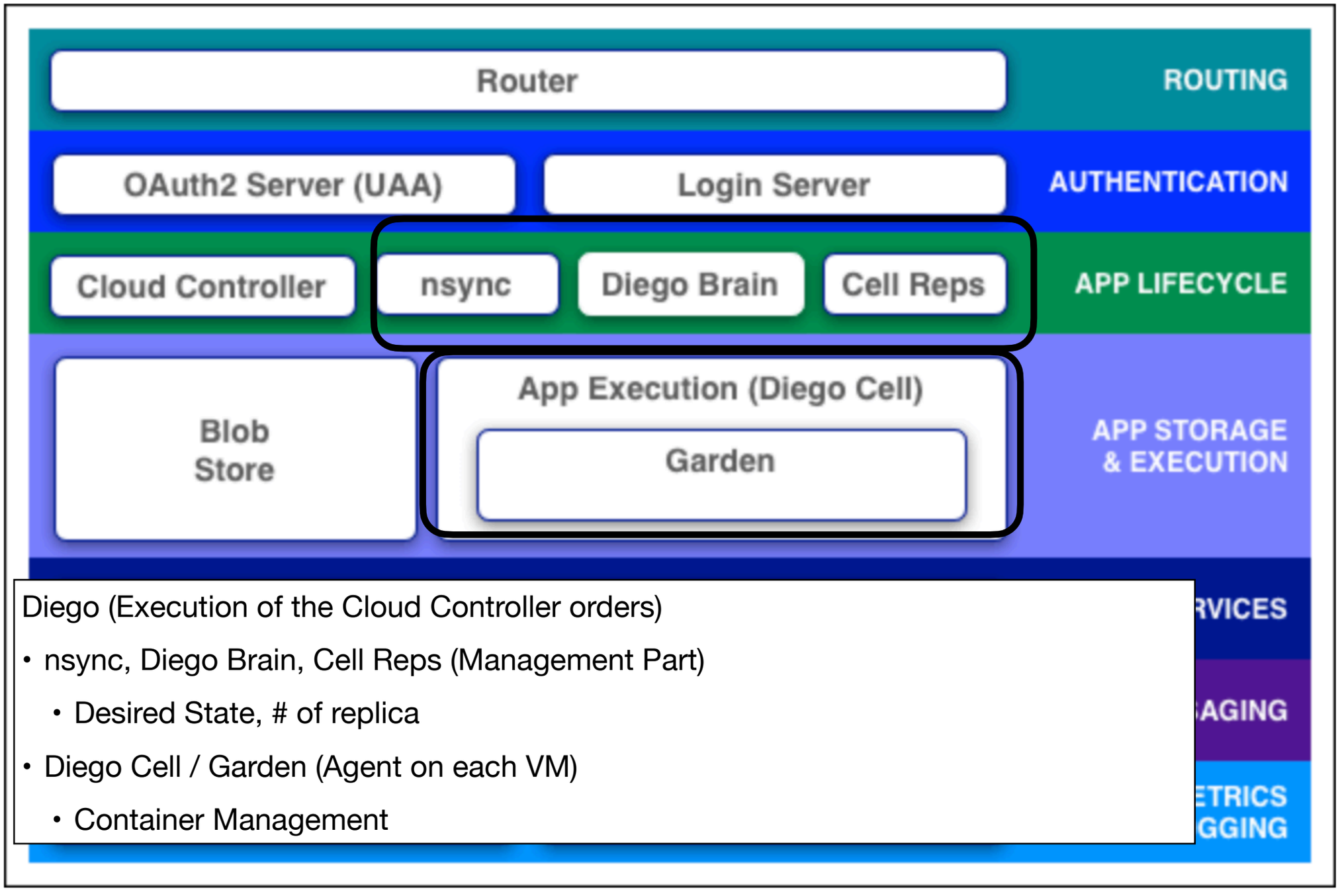
Cloud Foundry Architecture Component Overview



Cloud Controller : Deployment Engine

- Controls the Build and (staged) Deploy of a CF Push
- Directs the Deployment of the apps and the availability (desired state # of replicas) of the app

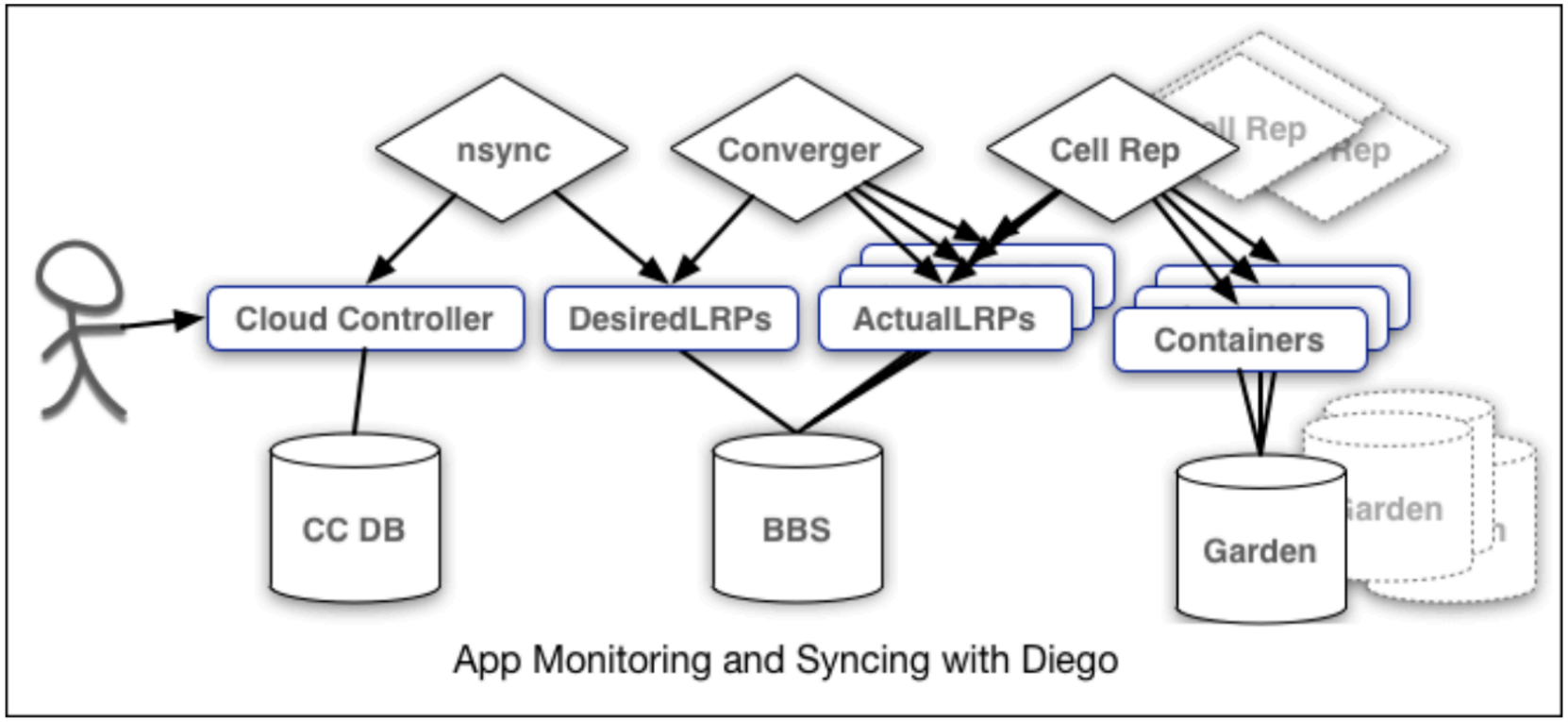
Cloud Foundry Architecture Component Overview



Diego (Execution of the Cloud Controller orders)

- nsync, Diego Brain, Cell Reps (Management Part)
  - Desired State, # of replica
- Diego Cell / Garden (Agent on each VM)
  - Container Management

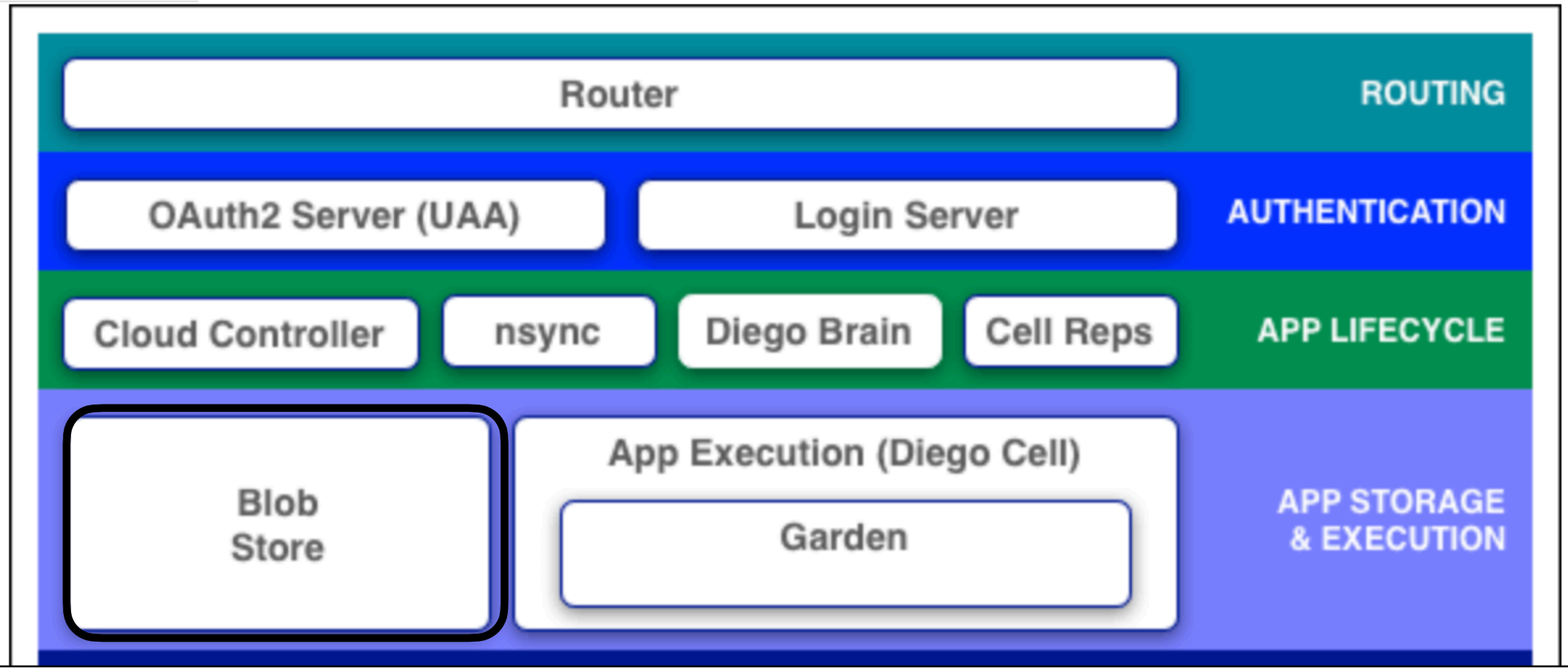
Desired State Management



App Monitoring and Syncing with Diego

- Cloud Controller directs the system wrt. # of active instances (LRP = Long running Process, normally a App instance)
- Converger (part of BBS) compares desired and actual # and either kills or launches new containers via the Diego cell) (BBS = Bulletin Board System is basically a data store for frequently updated and disposable data such as cell and application status, unallocated work, and heartbeat messages. The BBS stores data in MySQL)
- Cell Rep (one per VM) monitors the health of the containers
- CC DB BBS are persistent stores
- **Garden is the new container management backend it supports Linux Containers, runC (Open Container Initiative), Windows and Docker**
- Communications among the components via HTTP Messages

Cloud Foundry Architecture Component Overview



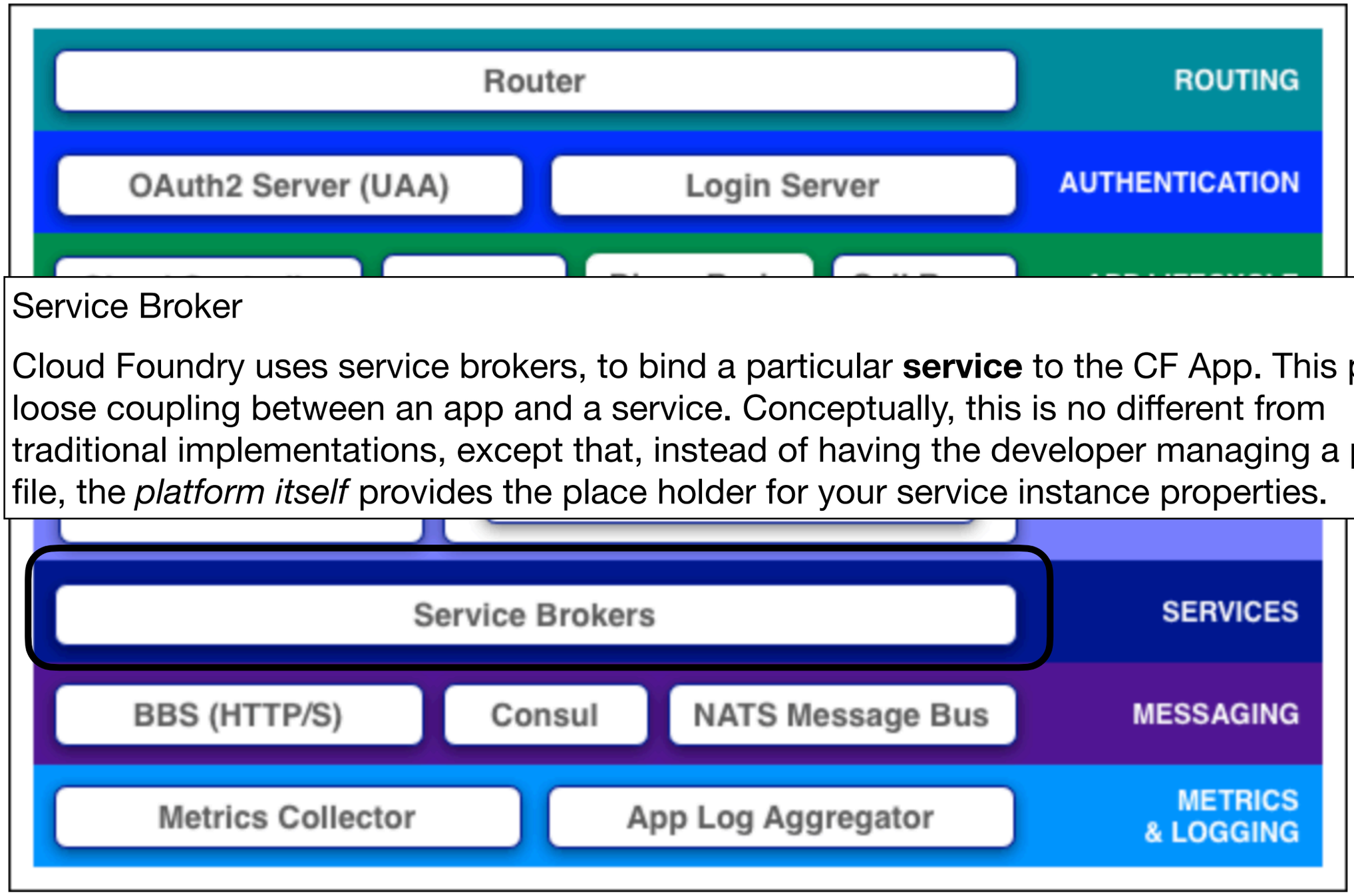
The blobstore is a repository for large binary files, the blobstore contains:

- Application code packages
- Buildpacks
- Droplets

You can configure the blobstore as either an internal server or an external S3 or S3-compatible endpoint.

- blob ? Distributed cloud applications often need to handle large data elements, also referred to as binary large objects (blob). Examples are virtual server images managed in an Elastic Infrastructure, pictures, or videos.

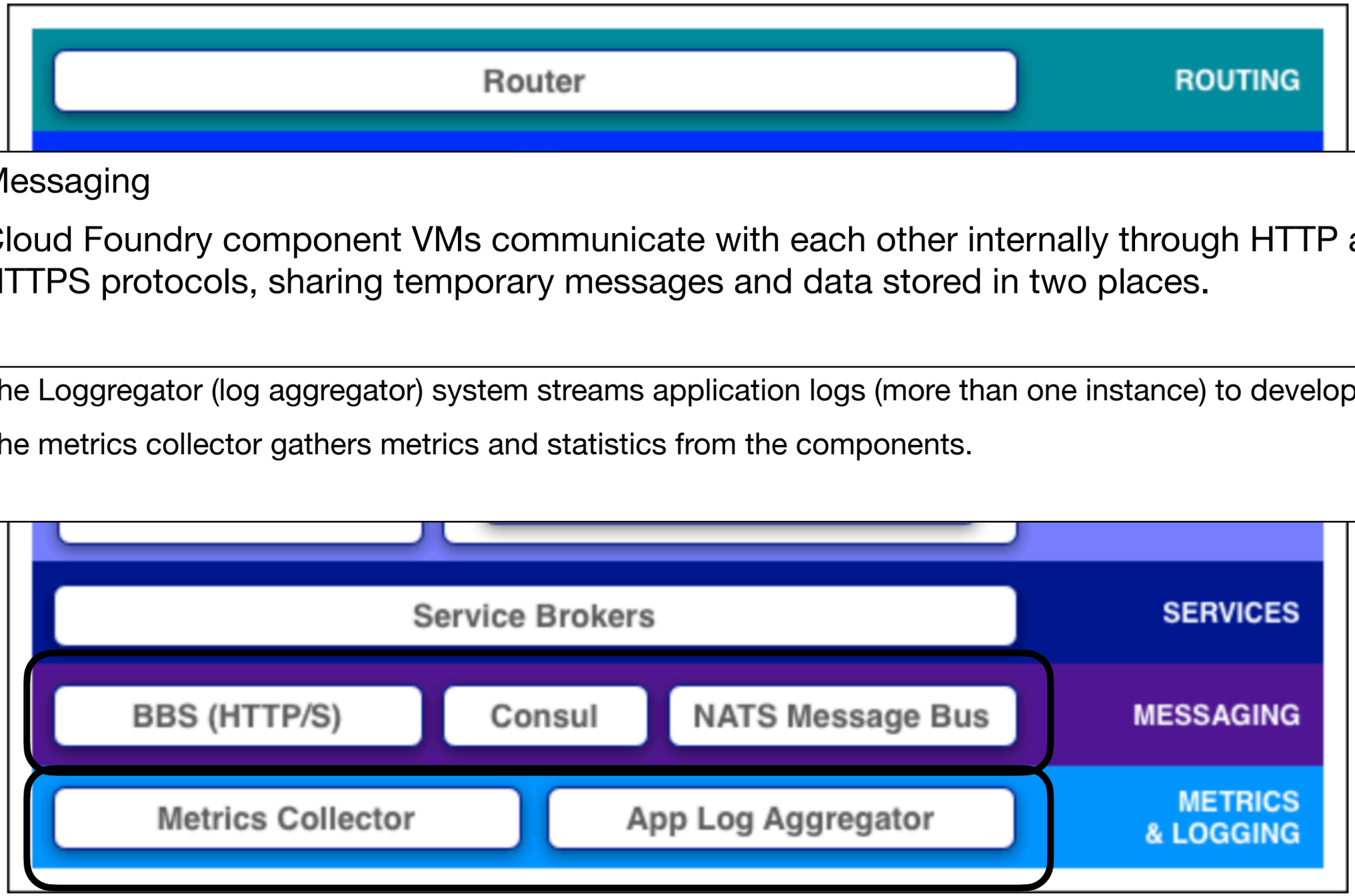
Cloud Foundry Architecture Component Overview



Service Broker

Cloud Foundry uses service brokers, to bind a particular **service** to the CF App. This permits loose coupling between an app and a service. Conceptually, this is no different from traditional implementations, except that, instead of having the developer managing a property file, the *platform itself* provides the place holder for your service instance properties.

Cloud Foundry Architecture Component Overview



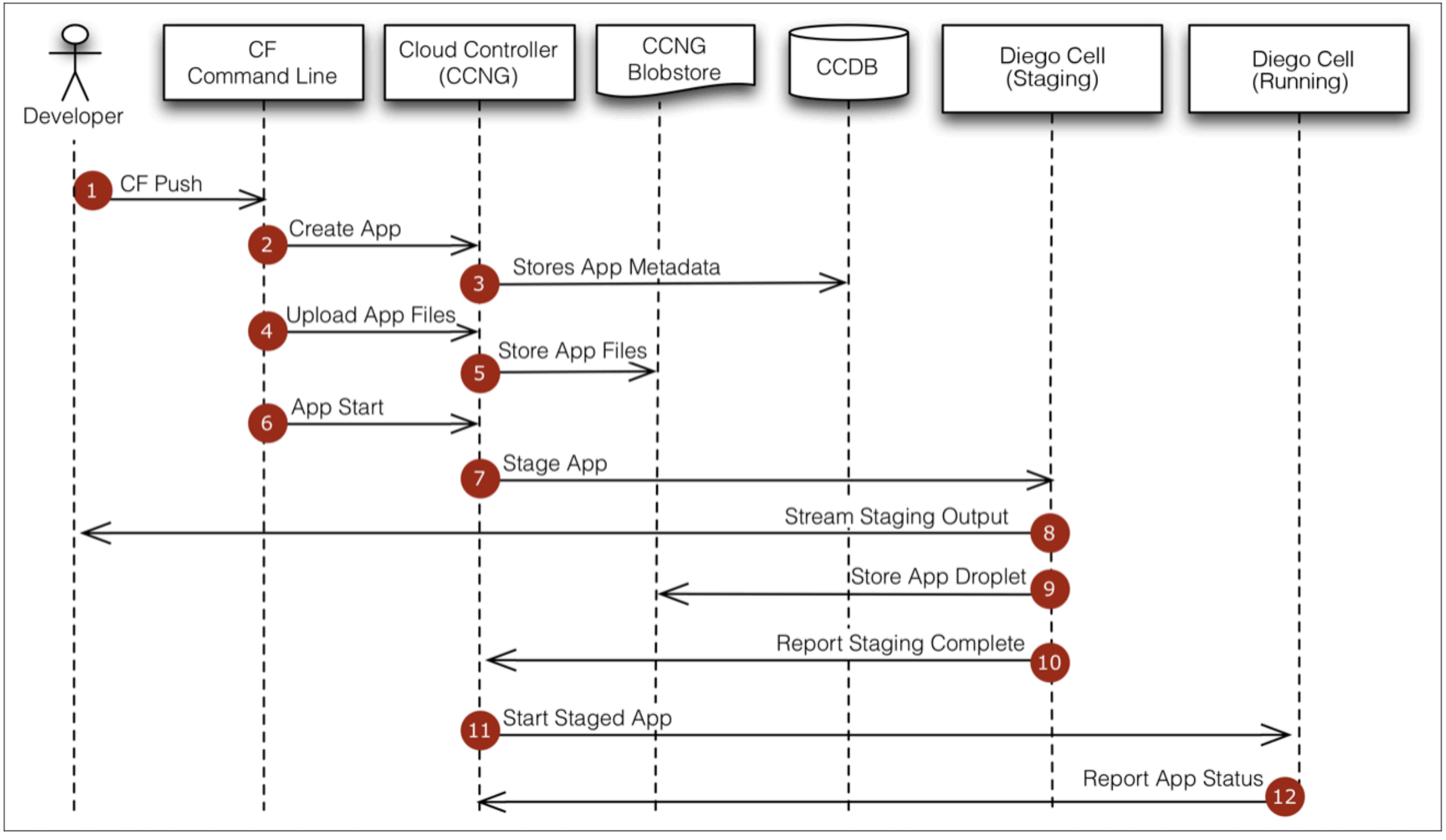
Messaging

Cloud Foundry component VMs communicate with each other internally through HTTP and HTTPS protocols, sharing temporary messages and data stored in two places.

The Loggregator (log aggregator) system streams application logs (more than one instance) to developers.

The metrics collector gathers metrics and statistics from the components.

The Code Deploy Flow



Cloud Foundry Manifest.YML

- Cloud Foundry does support docker images as well..
  - Finally CF push the Docker Image

```
Juergens-MBP:tippen juergenschneider$ bx cf push tippendocker --docker-image=juergenschneider/tippendocker:latest --no-manifest
Invoking 'cf push tippendocker --docker-image=juergenschneider/tippendocker:latest --no-manifest'...

Pushing app tippendocker to org DHBW Cloud Computing / space DHBW Exercises as jschnei1@lehre.dhbw-stuttgart.de...
Getting app info...
Creating app with these attributes...
+ name:           tippendocker
+ docker image:   juergenschneider/tippendocker:latest
  routes:
+   tippendocker.eu-gb.mybluemix.net

Creating app tippendocker...
Mapping routes...

Staging app and tracing logs...
   Cell 818ffad7-d39c-4424-90d2-dfa76f97df99 successfully created container for instance 5db9716a-c836-4787-9ecd-7a72324de45a
   Staging...
   Staging process started ...
   Staging process finished
   Exit status 0
   Staging Complete
   Cell 818ffad7-d39c-4424-90d2-dfa76f97df99 stopping instance 5db9716a-c836-4787-9ecd-7a72324de45a
   Cell 818ffad7-d39c-4424-90d2-dfa76f97df99 destroying container for instance 5db9716a-c836-4787-9ecd-7a72324de45a

Waiting for app to start...

name:              tippendocker
requested state:   started
instances:         1/1
usage:             1G x 1 instances
routes:            tippendocker.eu-gb.mybluemix.net
last uploaded:     Thu 25 Oct 15:01:53 CEST 2018
stack:             cflinuxfs2
docker image:      juergenschneider/tippendocker:latest
start command:     node myWebServer.js 8080

     state     since                  cpu    memory    disk      details
#0   running   2018-10-25T13:03:05Z   0.0%   0 of 1G   0 of 1G
```

Approaching SaaS
Writing Apps using AI Services

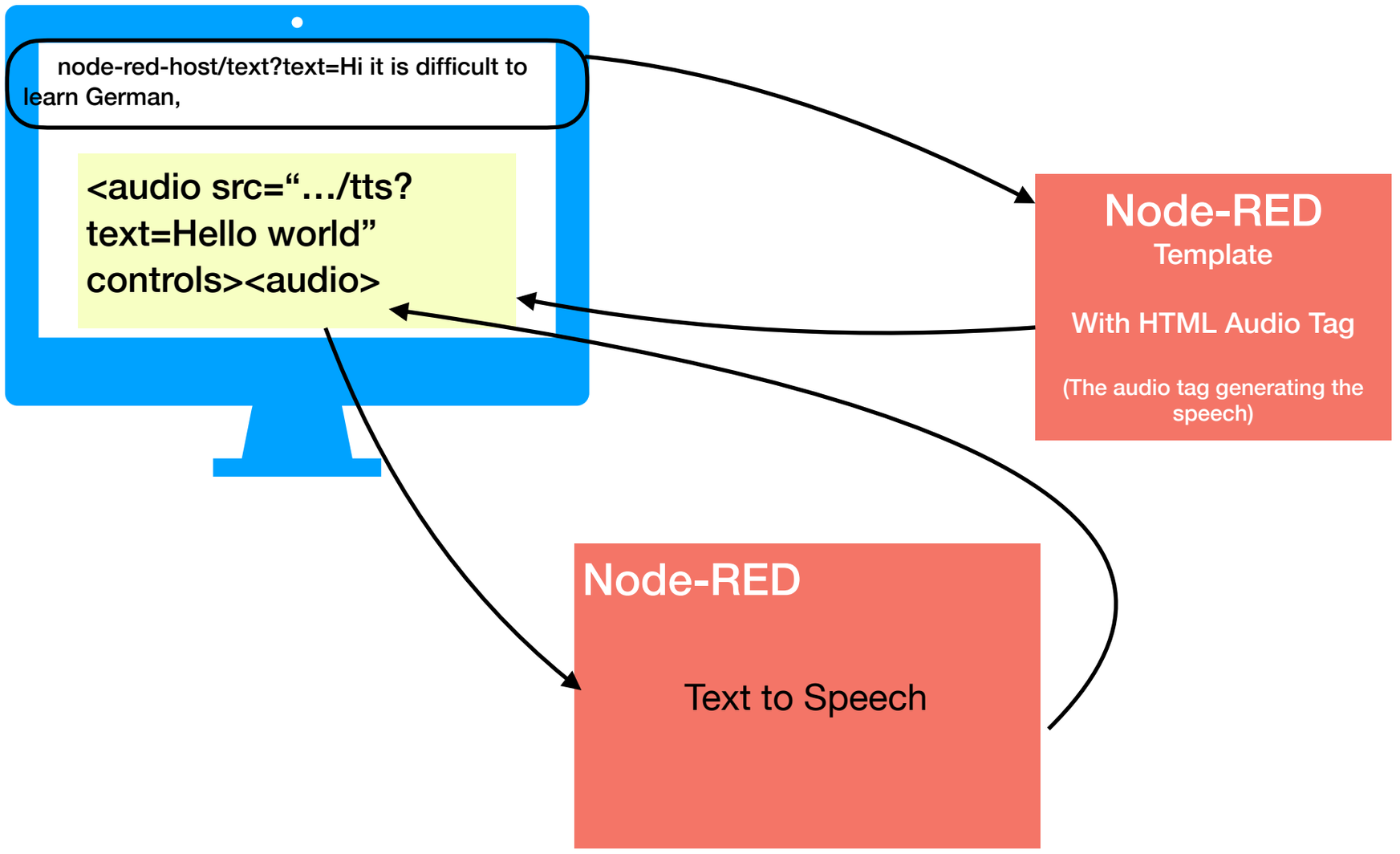
Node-Red Action Flow

- Often Apps are a flow of SaaS Services.
- Either your write them by yourself or use an action Modeller like Node-Red. It easily enables you to “**configure**” a Web Server (instead of writing your own Node.js).
- Node-Red is a node.js implementation with those modelling capabilities
- See Demo..

- A Node-RED Instance can be initiated
    - via Terraform in the bwCloud
        - Lösungen/bwCloud/nodered
- Admin. http://<bwcloud-ip:1880
- Services typically as http get with query string
    - Examples of flows in /internal/node-red

Node-Red Transaction Flow

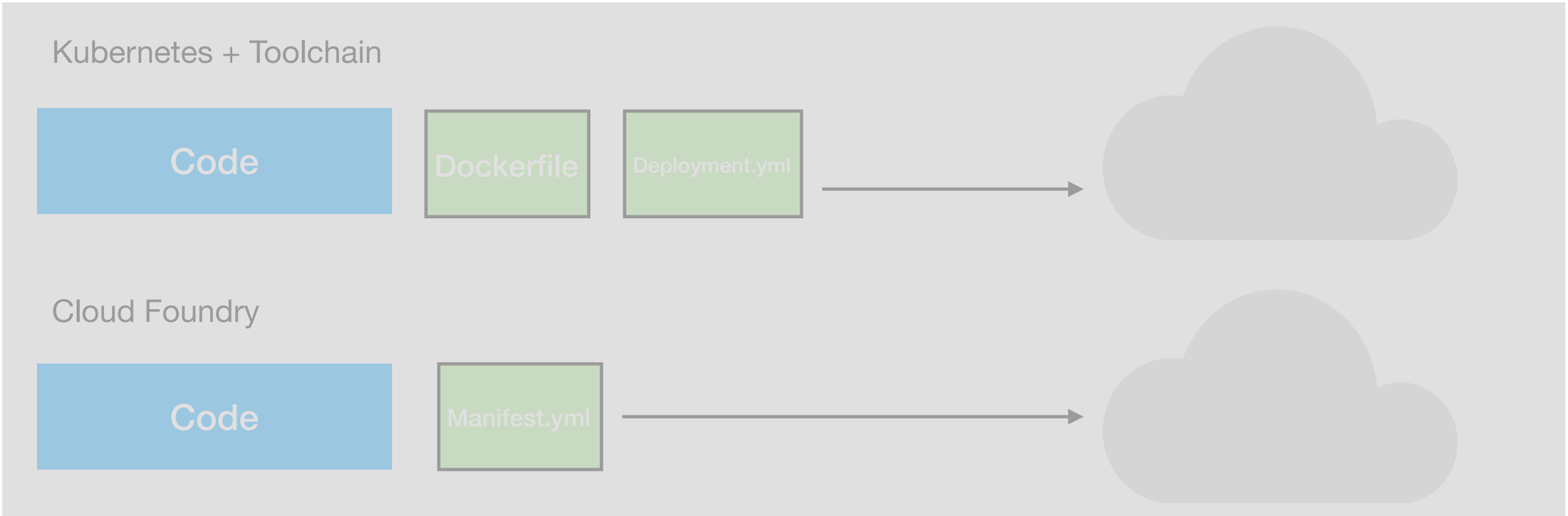
- Implement a simple Text to Speech use case via Node-Red
  - Flow Example in /internal/node-red
  - The HTML element which produces the the audio file must be changed to the new bwCloud Instance
  - The url is /text?text= and the text must be in English
  - There are two flow first ist to receive the the text to generate the HTML audio which it self will invoke the TTS Service

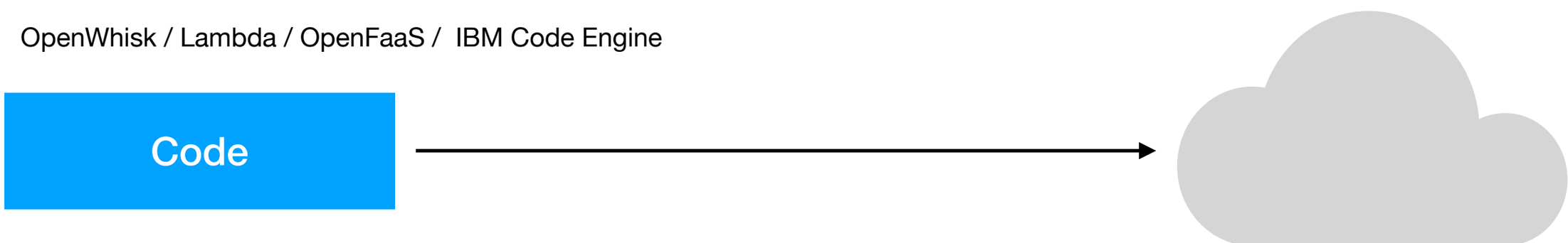


IaaS++(CaaS)　PaaS　PaaS++(FaaS)

Dev / Ops (User) provides it

Somebody else manage it





Event Driven Processing - Focus on Function

- Small and short program functions which are triggered by an event, execute that logic for that event and disappear.
  - Examples are:
    - Execute app logic in response to database triggers
    - Execute app logic in response to sensor data
    - Execute app logic in response of a HTTP API call
    - Execute app logic in response to scheduled tasks
  - For many of these scenarios, event-driven programming models are a better fit for **serverless architectures that can provide temporary resources on demand for intermittent tasks.**
- Cost
  - Applications deployed using a **serverless** architecture rely on a platform that provides the resources needed **on demand**, **at that very moment.** This obviously means that business logic execution can be mapped **to actual compute time used** (in milliseconds) rather than memory reserved for anticipated usage (by gigabyte per hour).
  - Good if those actions are ‘short’ (e.g. less than a minute) and the events happen infrequent or not equally distributed over time.
- Serverless is a bit misleading, better is to say the code (= function + MiniConfig) is in the cloud but the server (=Container) is built on demand (triggered by an event)

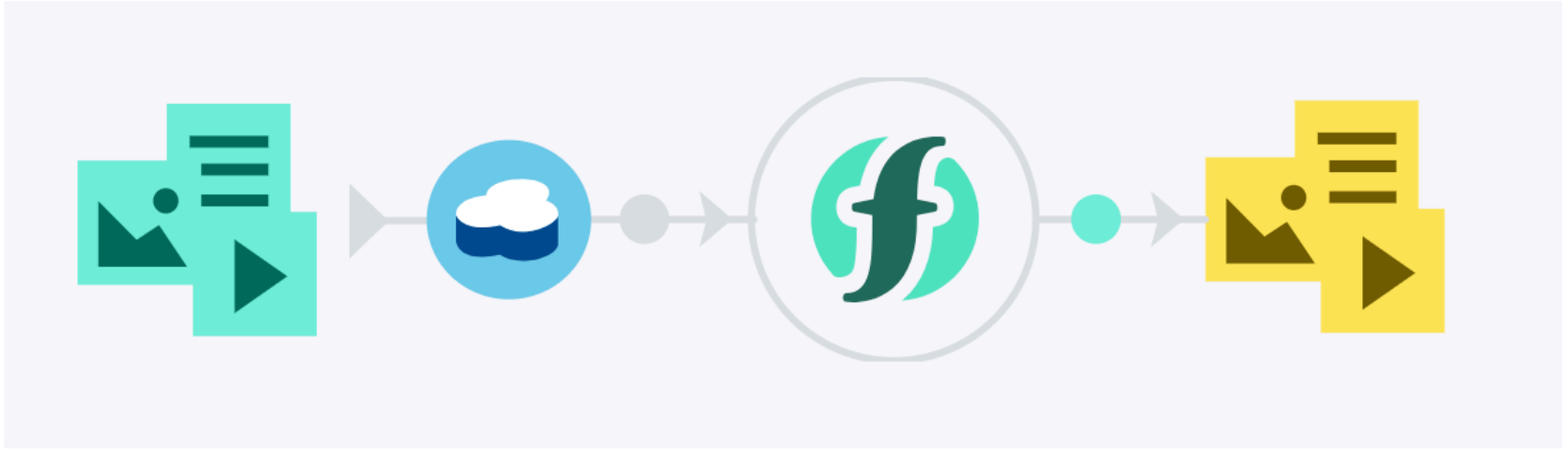
Some possible Scenarios

Serverless Backends

Expose application logic by implementing serverless microservices. Simply map your functions to well-defined API endpoints any client can call by making use of Web Actions or API Gateway integration.

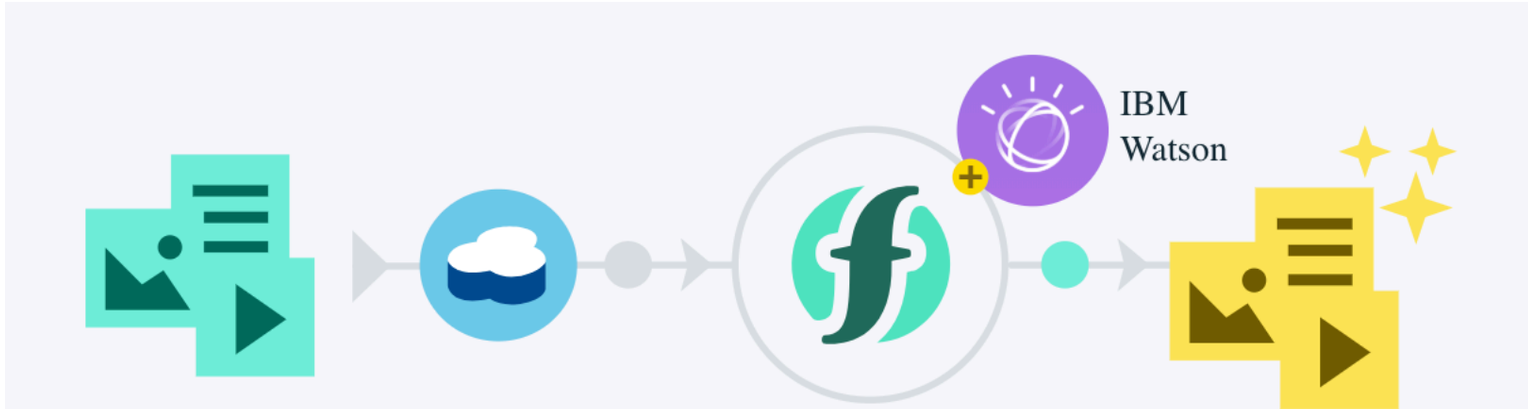
Data Processing

Execute code whenever data is updated in your datastore. Easily automate processes like audio normalization, image rotation, sharpening, noise reduction, thumbnail generation, or video transcoding.



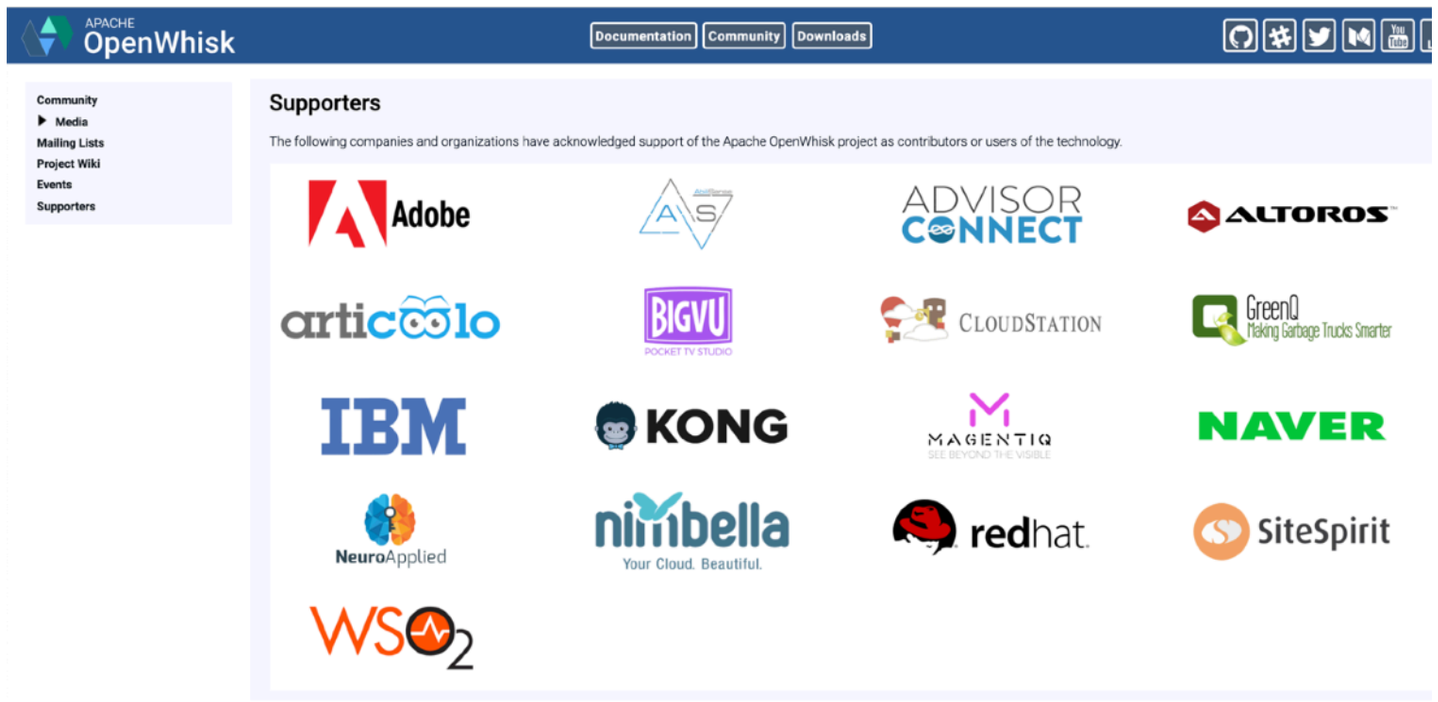
Cognitive Data Processing

Analyze data as soon as it becomes available. Let your function make use of powerful cognitive services like translation or image recognition to detect objects or people appearing in images or videos.



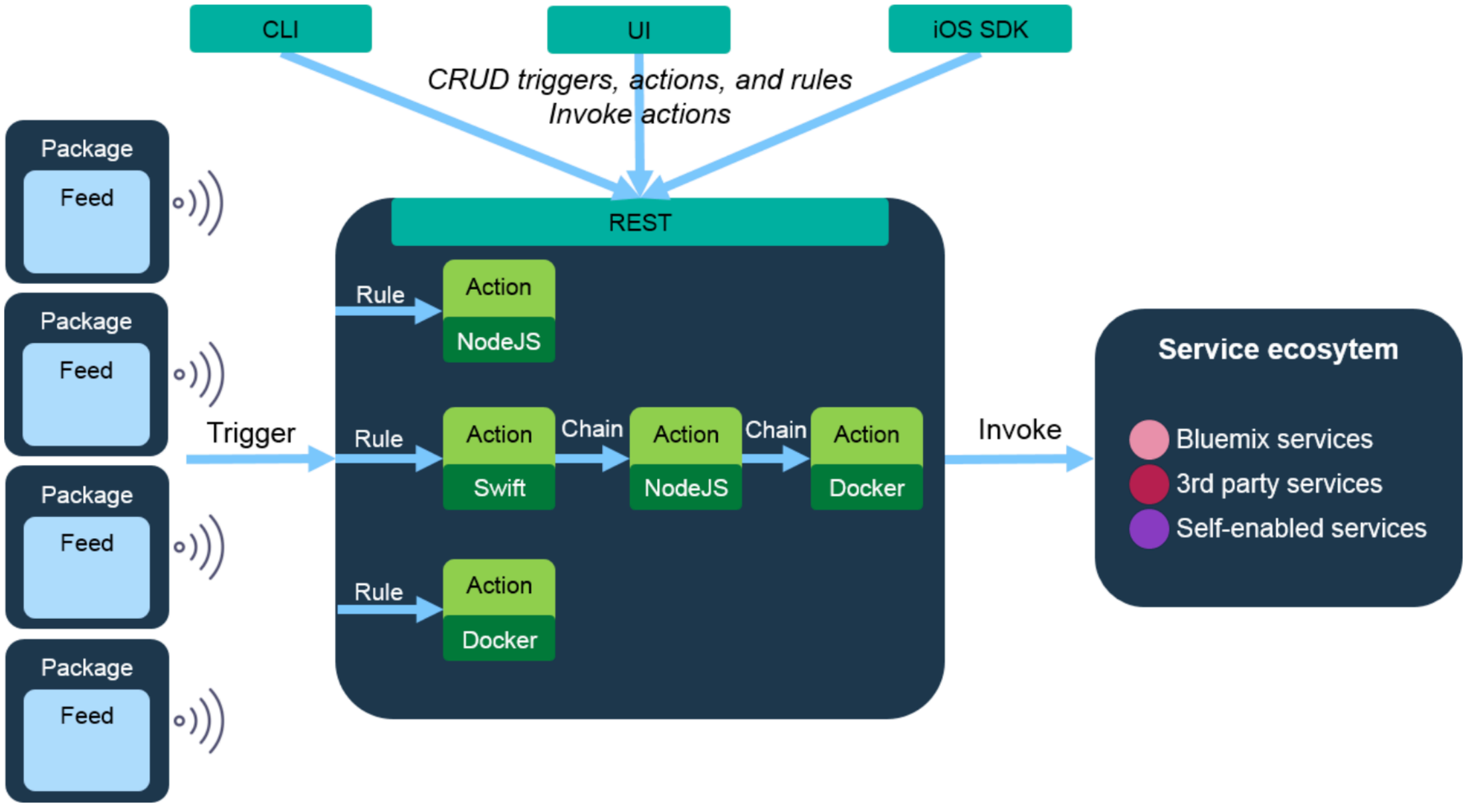
Serverless Runtimes or FaaS current Market

- AWS Lambda available since 2016
- Google Cloud Functions (as part of the Google App engine) since 2016
- Microsoft Azure Functions since 2016
- IBM Functions (on top of Open Source OpenWhisk) since 2017
- Oracle Cloud Platform
- Project Riff developed by Pivotal on top of Kubernetes

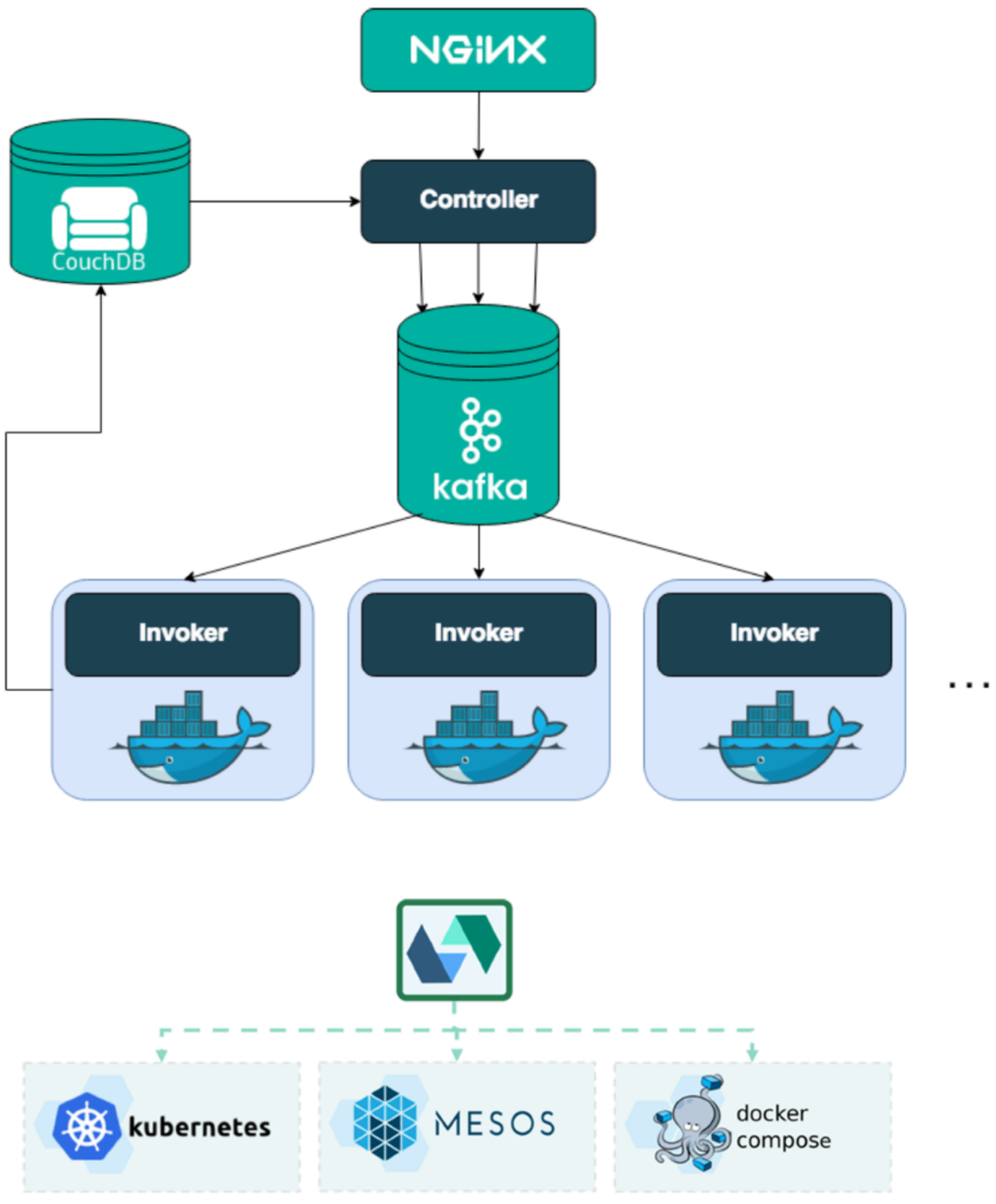


Apache OpenWhisk

https://openwhisk.apache.org/community.html



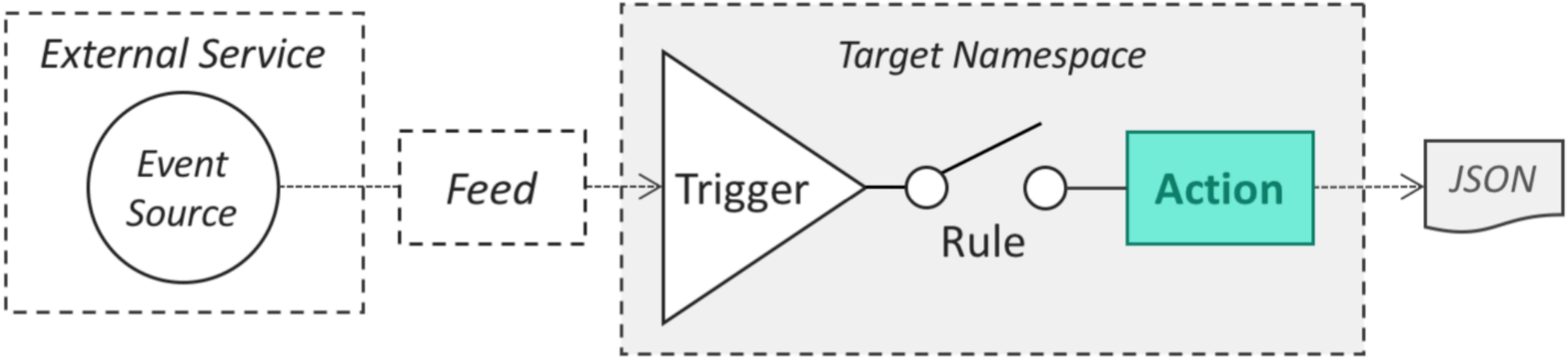
OpenWhisk High Level Flow



- NGINX (HTTP reverse Proxy Server)
  - It is mainly used for SSL termination and forwarding appropriate HTTP calls to the next component.
- Controller
  - Rest Processing
  - Authentication and Authorization
  - writes request to couch DB
  - Using Kafka message streaming
  - Assigns work among the invokers (invokers are in a service registry)
  - Returns the http caller (with an id)
  - (http invoker can query the results later with an additional HTTP Get id)
- Couch DB (Data Services)
  - Credentials
  - Packages (feeds/actions/rules)
  - Results of actions
- Consul
  - Key value store
- Invoker (Rest Executor)
  - Subscribes to requests from the controller.
  - Invoke the action
  - Each action in known docker
  - Put action results into Couch DB
- Kafka (Pub/Sub Streaming)
- A lot more….

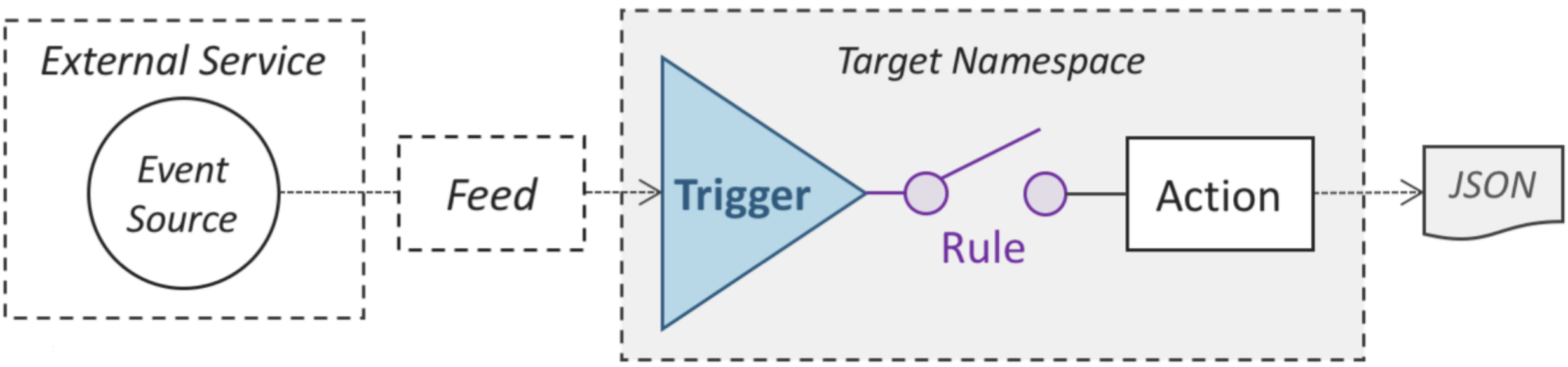
https://github.com/apache/openwhisk/blob/master/docs/about.md#how-openWhisk-works

Terminology in OpenWhisk



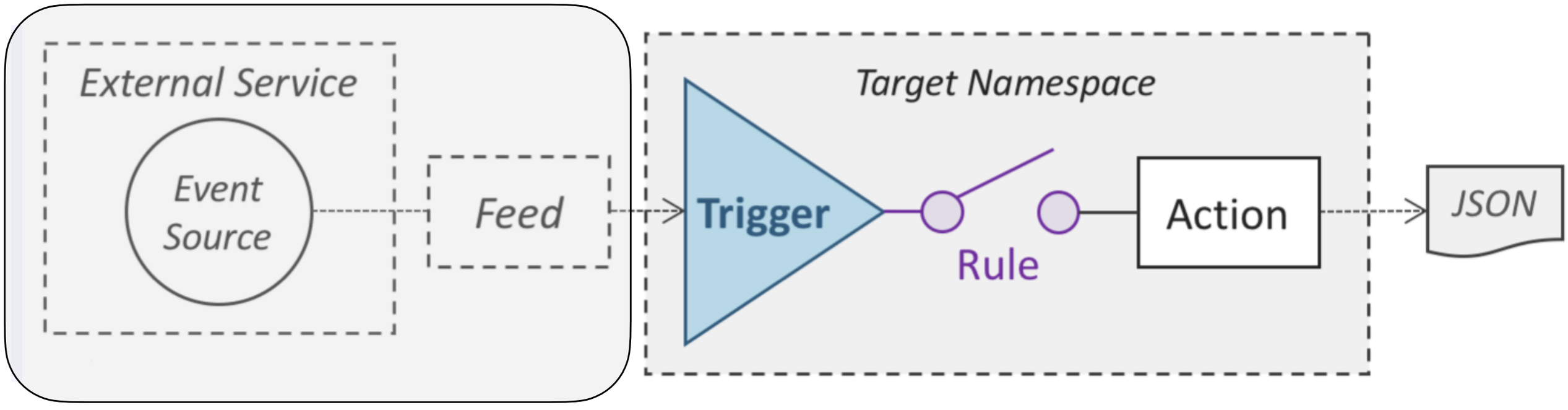
- Action
  - is a piece of code that performs one specific task. An action can be written in the languages described or a custom binary code embedded in a Docker container. **You provide your action to Cloud Functions either as source code or a Docker image**. (Limit is 40 MB)
  - An action performs work when it is directly invoked by using the Cloud Functions API, CLI or automatically respond to events (from an Cloud services) using a trigger.
- Sequence
  - A sequence is a chain of actions, invoked in order, where the output of one action is passed as input to the next action. This allows you to combine existing actions together for quick and easy re-use. A sequence can then be invoked just like an action, through a REST API or automatically in response to events.

Terminology in OpenWhisk (IBM Cloud Functions)



- Trigger
  - A trigger is a **declaration that you want to react to a certain type of event**
  - It describes the specific event (and the given parameters (key/value)) which should happen in order to fire the action
  - E.g. In **couch-db ABC** a document was added to **database XYX**
- Rule
  - Every time the trigger fires, the rule uses the trigger event as input and invokes the associated action.
  - With the appropriate set of rules, it's possible for a single trigger to invoke multiple actions, or for an action to be invoked from multiple triggers.

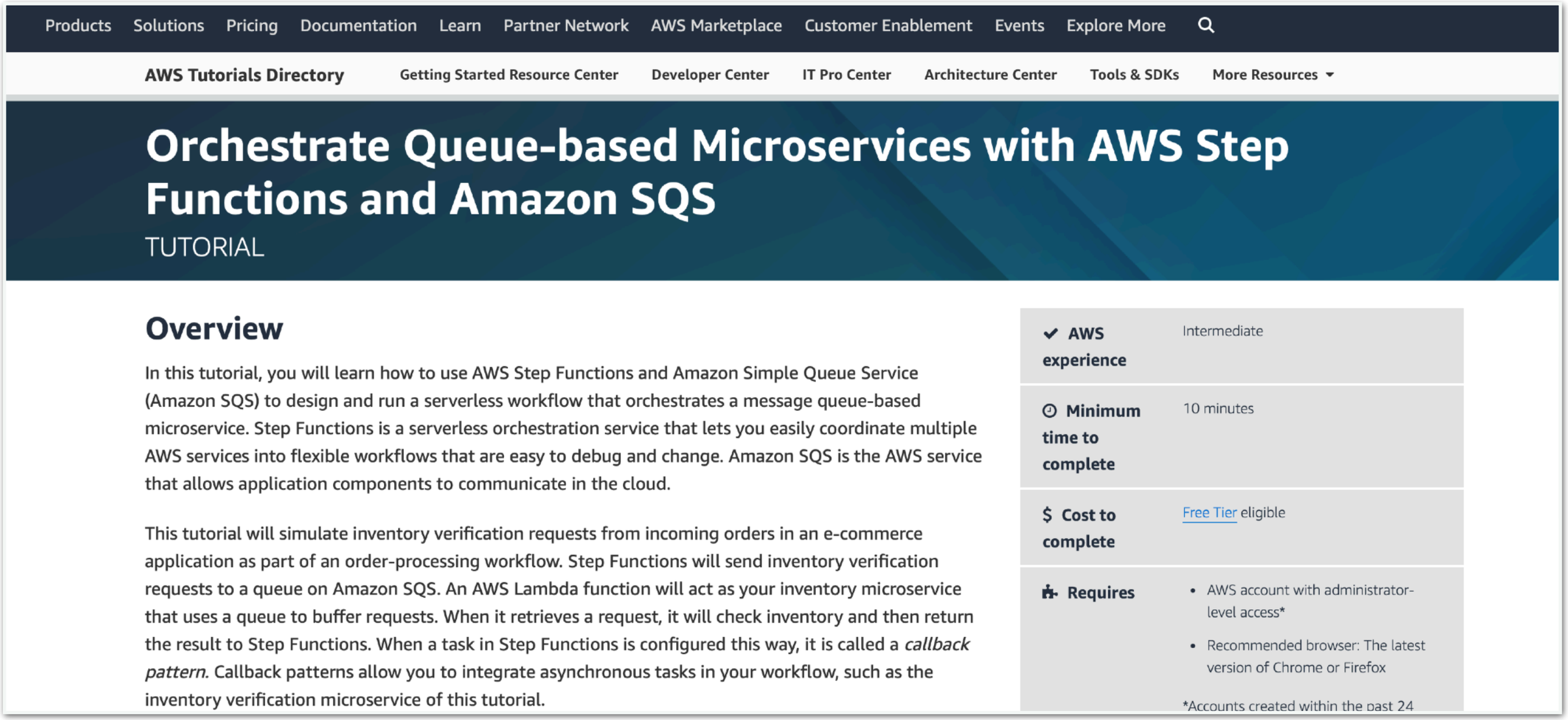
Terminology in OpenWhisk (IBM Cloud Functions)



- Feed
  - is a convenient way to configure an external event source to fire trigger events (e.g. web hooks). Firing an event is mostly a REST call
- Event Sources
  - These are services that generate events that often indicate changes in data or carry data themselves. Some examples of common Event Sources include:
    - Messages arriving on Message Queues
    - Changes in Databases
    - Changes in Document Stores
    - Website or Web Application interactions
    - Service APIs being invoked
    - IoT Frameworks forwarding device sensor data.

Workflows using Lambda

https://aws.amazon.com/tutorials/orchestrate-microservices-with-message-queues-on-step-functions/



PaaS on top of Kubernetes Recap

- Eine immer gestellte Frage ist, welche technische Details notwendig sind um einem Software Entwickler oder einer Software Entwicklerin alle notwendigen PaaS Optionen zu geben, die er oder sie braucht um die eigentlich Entwicklungsaufgabe effizient zu erledigen. Der andere Teil wird von einer Admin Role übernommen. Nenne zwei Beispiele für Konfigurationsoptionen die typischerweise von der Admin Rolle und zwei die typischerweise von der Entwicklung beschrieben werden sollte.
- Positioniere Serverless zu Function as Service)
- Knative unterscheidet zwischen Applications, Jobs und Funktionen. Welche unterschiedliche Scaling Profile haben diese Workloads. (Einfache Beschreibung)
- Buildpacks (welches nun auch in einer PaaS Integration mit Kubernetes zum Einsatz kommen) ist ein Satz von Scripts die in 4 Phasen (detect, build, prepare and run) den zur Verfügung gestellten Code in der PaaS Cloud zum Laufen bringt. Welche Phasen gibt es und was tuen sie (grob am Beispiel von Node.js.)
- Knative definiert den Begriff Revision. Was steck hinter einer Revision
- Unter dem Begriff Serverless versteht man, daß ready to run code (=images) nur im Bedarfsfalle aktiviert wird.
- Der große Vorteil (den die Cloud Provider postulieren) ist, daß nur bei Ausführung des Codes (aktiver Container) auch abgerechnet wird.
- Der Nachteil ist die Performance wenn in gewissen Situationen erst der Container gestartet werden muss und die App sich ggfls. noch initialisiert .. das ist spürbar. Der Gegensatz ist die Anwendung hat einen Server ist aktiv und wartet auf eine Anfrage.
- Neben der Knative und OpenFass Implementierung gibt es noch andere Implementierung von FaaS. Nenne 2 zwei. (Openwisk, AWS mit Lambda und Azure Functions)
- Typische Events für Funktionen sind DB Updates/Changes, IOT Data Arrivals (Kafka), Cron Triggers. HTTP Web Service Endpoints.

Referenzen

- Cloud Foundry
  - Steffen Uhlig, IBM. https://ws.uhlig.it
  - https://docs.cloudfoundry.org/concepts/architecture/
  - https://cloudacademy.com/blog/cloud-foundry-components/
  - Vorlesungsscript, Prof. Dr. Ulrike Steffens, Hochschule für Angewandte Wissenschaften Hamburg
  - https://docs.cloudfoundry.org/buildpacks/understand-buildpacks.html
  -
- OpenWisk
  - https://developer.ibm.com/opentech/2016/09/06/what-makes-serverless-attractive/
  - https://github.com/apache/incubator-openwhisk/blob/master/docs/about.md
  -