CLOUD COMPUTING SYSTEMS

Lecture 5

Nuno Preguiça

(nuno.preguica_at_fct.unl.pt)

OUTLINE

Serverless

- Function-as-a-Service

Services for micro-service based applications

OUTLINE

Serverless

- **Function-as-a-Service**

Services for micro-service based applications

PROBLEMS

Some services can be coded in only a few lines of code.

Some services are called infrequently.

The load of some services can change very fast.

Are these services a good fit for something like Azure App Service / Google App Engine? Why?

Having a machine / server running these services all the time is an overkill.

SOLUTION: CLOUD FUNCTIONS

Cloud functions are small pieces of code, written in a high-level language.

A cloud function is invoked when some event occurs – such as loading an image into cloud storage, receiving a read from a sensor, etc.

The serverless system is responsible for the provisioning and administration tasks: instance selection, scaling, deployment, fault tolerance, monitoring, logging, security patches, and so on.

Amazon introduced the first service with cloud functions in 2015: AWS Lambda.

This model is called: function as a service (FaaS).

SERVERLESS OR NOT...

Serverless computing is still done in servers. ☺

.. but server provisioning and administration tasks are done by the cloud provider.

A serverless service must scale automatically with no need for explicit provisioning, and be billed based on usage.

Other serverless frameworks include BaaS (Backend as a Service) offerings.

Serverless computing = FaaS + BaaS.

Key difference between serverless and "serverful"

Decoupled computation and storage. The storage and computation scale separately and are provisioned and priced independently. In general, the storage is provided by a separate cloud service and the computation is stateless.

Automatic resource allocation. Instead of requesting resources, the user provides a piece of code and the cloud automatically provisions resources to execute that code.

Paying in proportion to resources used instead of for resources allocated. Billing is by some dimension associated with the execution, such as execution time, rather than by a dimension of the base cloud platform, such as size and number of VMs allocated.

What makes FaaS possible

FaaS must be provisioned fast and offer strong security isolation.

Virtual machines and containers are the key to make FaaS possible, making multi-tenant hardware sharing possible.

For making the creation of the execution environments for running function fast, several optimization are users:

- "warm pool" of VM instances that need only be assigned to a tenant;
- "active pool" of instances that have been used to run a function before and are maintained to serve future invocations.

What are the possible uses of FaaS?

Ideas, please.

FaaS example: image processing

- Goal: create thumbnails for images in the system.
- Goal: replicate the image to some other system.
- When an image is added to the storage, run a cloud function to create the thumbnail.

- Advantages when compared with a traditional solution?

FaaS example: video encoding

- Goal: parallelize the slow tasks of video encoding.
- The video encoding can be parallelized (in large part). It is possible to use FaaS to scale to large numbers of parallel functions.

- Advantages when compared with a solution based on having server to run the task?
- 60x faster, 6x cheaper versus VM instances [1].

AZURE FUNCTIONS

Azure Functions is a solution for easily running small pieces of code, or "functions," in the cloud.

You can write just the code you need for the problem at hand, without worrying about the complete application or the infrastructure to run it.

Functions can make development even more productive, and you can use your development language of choice, such as C#, Java, JavaScript, PowerShell, and Python.

Pay only for the time the code runs – Azure scales as needed.

Azure Functions let users develop serverless applications on Microsoft Azure.

AZURE FUNCTIONS TRIGGERS

HTTPTrigger – function executed on an HTTP request.

TimerTrigger – execute cleanup or other batch tasks on a predefined schedule.

CosmosDBTrigger - Process Azure Cosmos DB documents when they are added or updated.

BlobTrigger - Process Azure Storage blobs when they are added to containers.

EventHubTrigger - Respond to events delivered to an Azure Event Hub, such as in application instrumentation, and internet-of-things (IoT) scenarios.

...

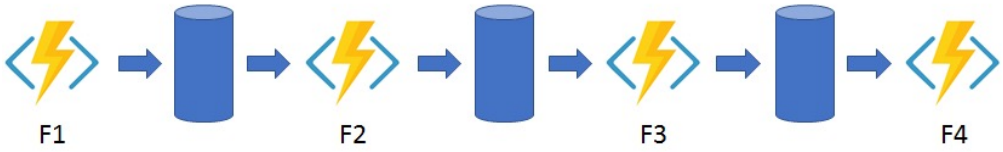
AZURE DURABLE FUNCTIONS

Durable Functions is an extension of Azure Functions for writing stateful functions.

Azure function can have stateful entities. The system manages state, checkpoints, and restarts.

DURABLE FUNCTIONS: EXAMPLES

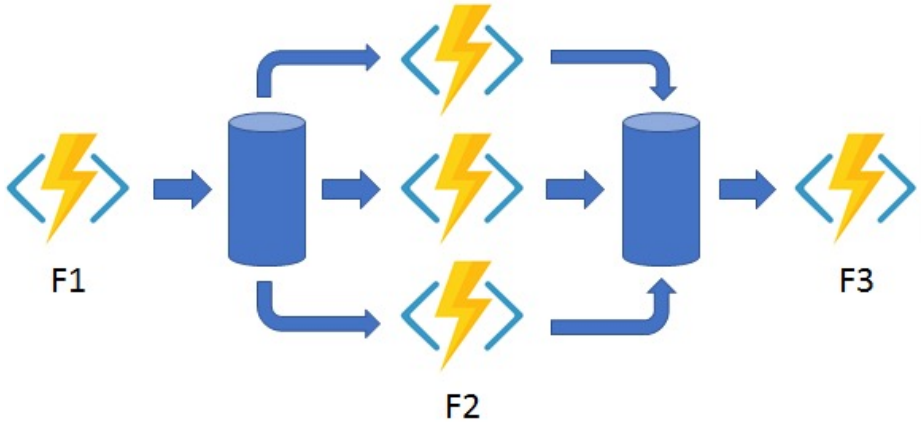
Execute a workflow/sequence of functions.




```
const df = require("durable-functions");

module.exports = df.orchestrator(function*(context) {
    try {
        const x = yield context.df.callActivity("F1");
        const y = yield context.df.callActivity("F2", x);
        const z = yield context.df.callActivity("F3", y);
        return    yield context.df.callActivity("F4", z);
    } catch (error) {
        // Error handling or compensation goes here.
    }
});
```

DURABLE FUNCTIONS: EXAMPLES (2)

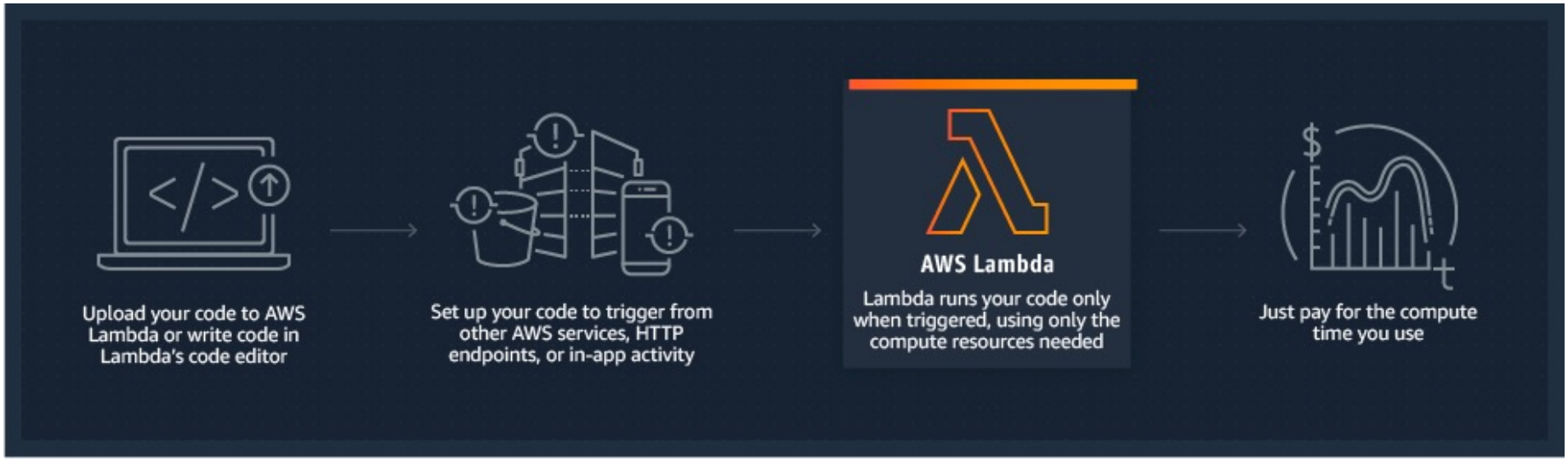
Execute several functions in parallel




```
const df = require("durable-functions");

module.exports = df.orchestrator(function*(context) {
    const parallelTasks = [];

    // Get a list of N work items to process in parallel.
    const workBatch = yield context.df.callActivity("F1");
    for (let i = 0; i < workBatch.length; i++) {
        parallelTasks.push(context.df.callActivity("F2", workBatch[i]));
    }

    yield context.df.Task.all(parallelTasks);

    // Aggregate all N outputs and send the result to F3.
    const sum = parallelTasks.reduce((prev, curr) => prev + curr, 0);
    yield context.df.callActivity("F3", sum);
});
```

AWS LAMBDA

"AWS Lambda lets you run code without provisioning or managing servers. You pay only for the compute time you consume - there is no charge when your code is not running."

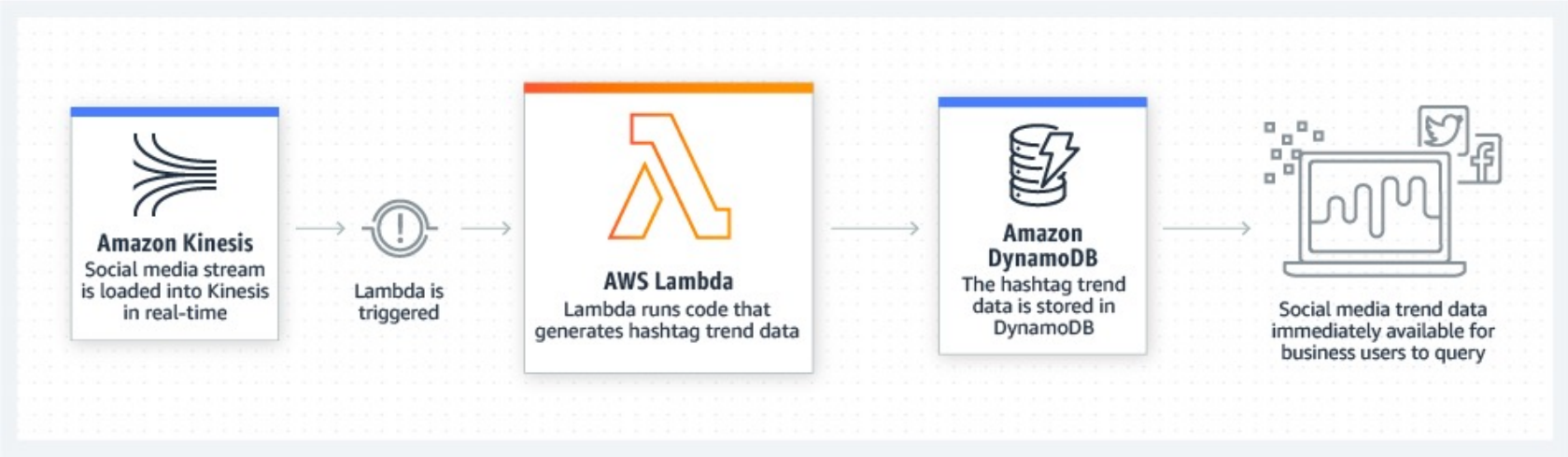


From: https://aws.amazon.com/lambda/

USE CASES

Realtime file processing – execute

Realtime stream processing

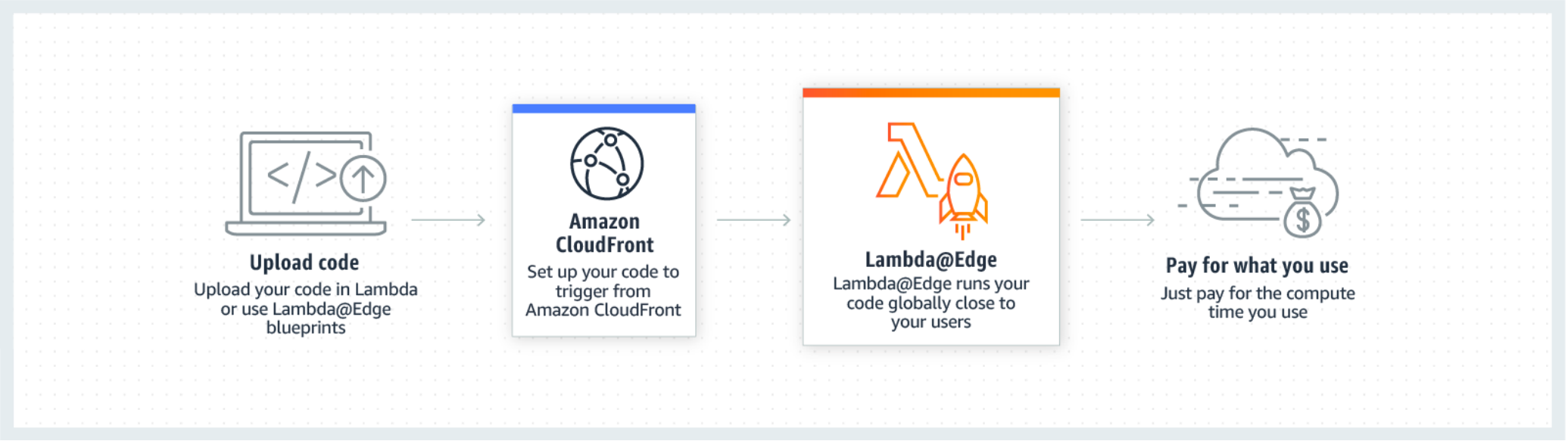


Extract, transform, load

LAMBDA@EDGE

Lambda@Edge is a service that allows to run Lambda functions in the Amazon points-of-presence (CDN).

As other FaaS services, clients pay only the compute time they consume.



USE CASES

Ideas?

Use case: Website Security and Privacy

Add HTTP security headers on all origin responses.

"This helps improve security and privacy for your users and content providers, while using CloudFront to deliver the content at low latencies."

Use case: Dynamic Web Application at the Edge

Run web applications at the edge, instead of running them in the data centers. Requires access to other services for storing data.

Use case: Real-time Image Transformation

Transform images on the fly to customize users' experience based on the characteristics of the users' devices.

For example, it is possible to resize images based on the viewer's device type – mobile, desktop, or tablet.

CDN can cache the transformed images.

FaaS system challenges: Inadequate storage for fine-grained operations

Difficult to support applications that have fine-grained state sharing needs (due to stateless nature).

Limitation of current storage services.

- BLOB storage services are highly scalable and provide inexpensive long-term object storage, but exhibit high access costs and high access latencies.
- Cloud databases are expensive and can take a long time to scale up.

Relying to caching services, such as Redis, is a good solution.

FaaS system challenges: Poor performance for standard communication patterns

Broadcast, aggregation, and shuffle are some of the most common communication primitives in distributed systems.

- These operations are commonly employed by applications such as machine learning training and big data analytics.
- With cloud functions the complexity of these tasks increases, as the number of tasks also increases.

FaaS system challenges: Predictable Performance

The delays incurred when starting new instances can be high for some applications and are unpredictable

Sources of unpredictability:

- the time it takes to start a cloud function;
- the time it takes to initialize the software environment of the function, e.g., load Python libraries;
- application-specific initialization in user code;
- whether a new VM/container needs to be created or not.

OUTLINE

Serverless

- Function-as-a-Service

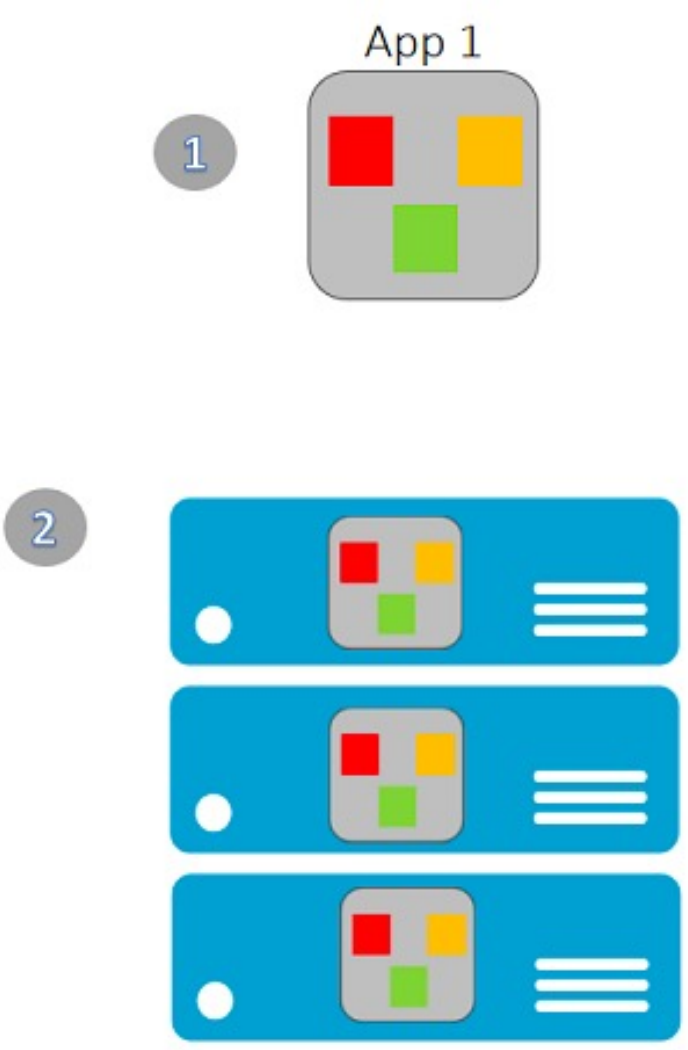
Services for micro-service based applications

FROM MONOLITHIC APPLICATIONS...

In a traditional monolithic application, all functionality is in the same application.

The application is scaled by running the whole application at multiple servers.

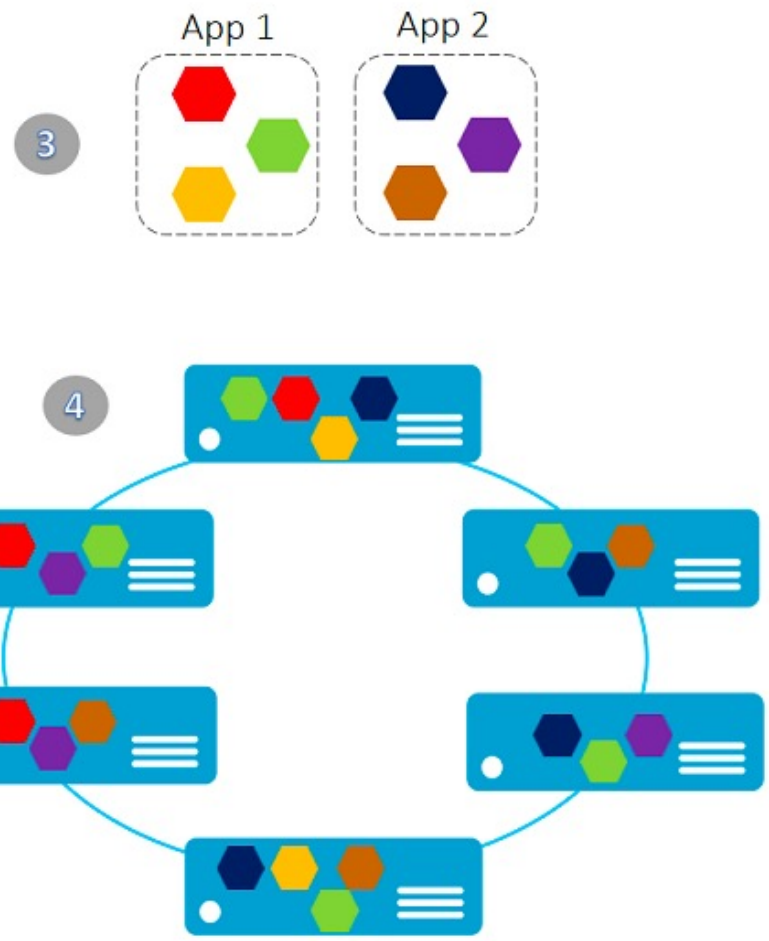
- E.g. in a web application, the application server can be run in multiple machines.



.. TO MICRO-SERVICES

In a micro-service application, different functionalities are separated into smaller services.

A microservice-based application scales out by deploying each service independently, creating instances of these services across servers/virtual machines/containers.



What is a micro-service?

Some desirable properties:

- Consists of code, and optionally state, both of which are independently versioned, deployed, and scaled.
- Interacts with other microservices over well-defined interfaces and protocols.
- Has a unique name (URL) that is used to resolve their location.
- Can be written in any programming language.
- Should remain consistent and available in the presence of failures.

POTENTIAL ADVANTAGES OF MICRO-SERVICES

- Small Modules – easier to program and maintain.
- Easier Evolution and Process Adaption – easier to evolve the system and adapt new technologies, as it is possible to adapt only some of the micro-services.
- Independent scaling – possible to scale each micro-service independently.
- Independent faults – a fault in a micro-service does not affect other micro-services.

POTENTIAL CHALLENGES FOR MICRO-SERVICES

- Monitoring and management – Necessary to manage a potential large number of micro-services, continuously monitoring the state of the services.
- Configuration and Deployment – Need to keep the configuration for all micro-services and automate its deployment.
- Application support – what services exist for simplifying inter-service communication and coordination?
- Debugging and testing – Debugging and testing become very complex, as a request leads to multiple requests; difficult to test micro-services independently.

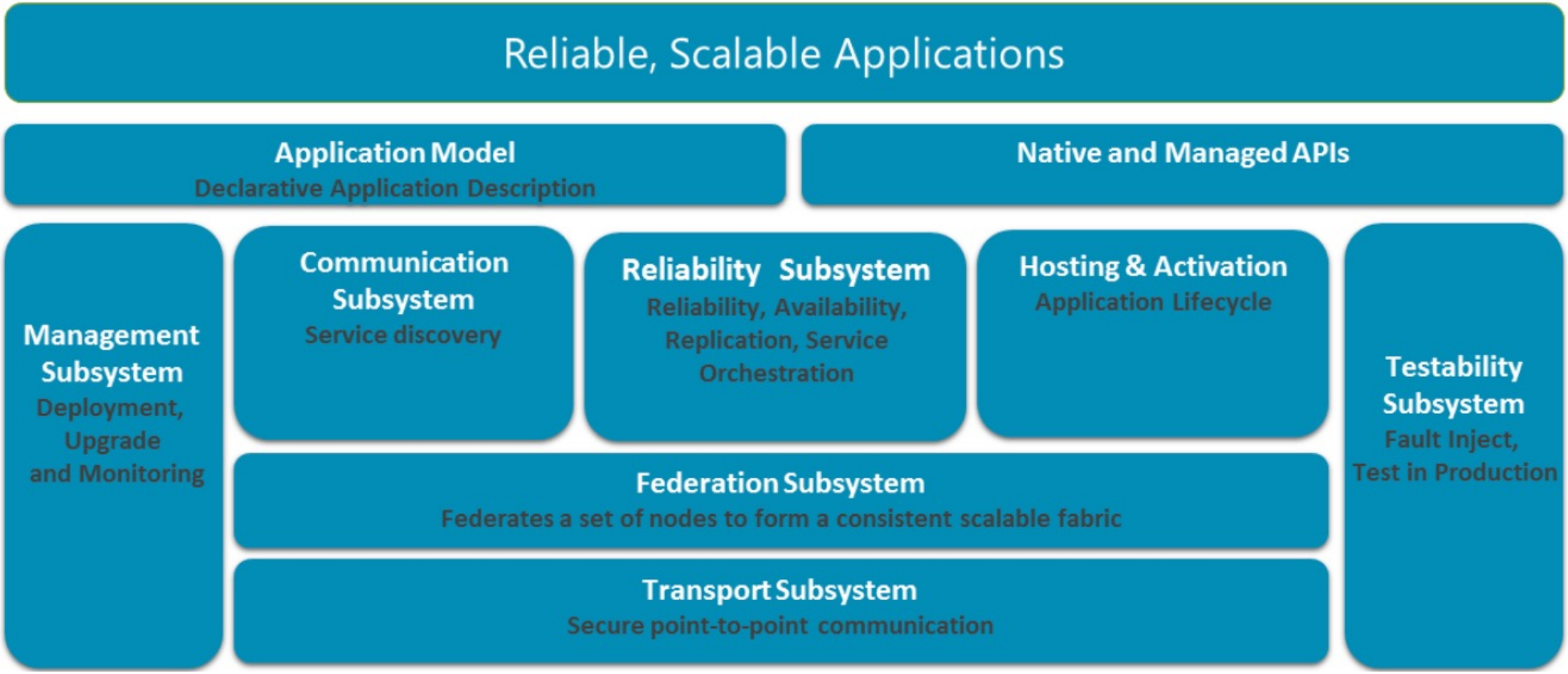
SUPPORT FOR MICRO-SERVICES

Building applications based on micro-services requires supporting services for addressing the following challenges:

- How to manage micro-services?
- How to make micro-services reliable?
- How to find a micro-service?
- How to communicate with a micro-service?

Azure Service Fabric

Service Fabric is built with layered subsystems.



TRANSPORT SUBSYSTEM

The ***Transport Subsystem*** is used for communication within a cluster and with clients.

Supports one-way and request-reply communication patterns.

This subsystem is used internally by Service Fabric and is not directly accessible by application.

FEDERATION SUBSYSTEM

The ***Federation Subsystem*** serves for managing a cluster. It provides the distributed systems primitives needed by the other subsystems - failure detection, leader election, and consistent routing.

The federation subsystem is built on top of DHT.

RELIABILITY SUBSYSTEM

The ***Reliability Subsystem*** provides the mechanism to make the state of a Service Fabric service highly available.

The **Replicator** replicates the state of micro-services – default replicator uses primary/backup replication.

The **Failover Manager** reconfigures the cluster and replicas to distributed load and maintain availability.

The **Resource Manager** manages resources to guarantee that micro-services are operational.

MANAGEMENT SUBSYSTEM

The ***Management subsystem*** provides end-to-end service and application lifecycle management.

Cluster Manager: The cluster manager manages the lifecycle of the applications from provision to de-provision.

Health Manager: This service enables health monitoring of applications, services, and cluster entities.

Image Store: This service provides storage and distribution of the application binaries.

COMMUNICATION SUBSYSTEM

The ***Communication subsystem*** provides reliable messaging within the cluster and service discovery through the Naming service. The Naming service resolves service names to a location in the cluster.

Using the Naming service, clients can securely communicate with any node in the cluster to resolve a service name and retrieve service metadata.

PROGRAMMING MODELS (SELECTED)

Reliable services

Reliable Services is a light-weight framework for writing services that integrate with the Service Fabric platform and benefit from the full set of platform features.

Reliable actors

Built on top of Reliable Services, the Reliable Actor framework implements the Virtual Actor pattern, based on the actor design pattern.

The Reliable Actor framework uses independent units of compute and state with single-threaded execution called actors.

Micro-services and Serverless

Micro-services are an old idea, but creating application based on micro-services was complex.

Renewed interest with the creation of supporting services/platforms. Success story of Netflix.

E.g.: Netflix OSS services for micro-services.

https://netflix.github.io/

Serverless is a newer idea. Allows to realize some of the goals of micro-services with a more incremental way.

To know more

https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-overview-microservices

https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-architecture

[1] https://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-3.html

ACKNOWLEDGMENTS

Some text and images from Microsoft Azure online documentation and AWS online documentation.