Distributed Systems CS6421 **Cloud Application Architectures**

Prof. Tim Wood and Prof. Roozbeh Haghnazar

Includes material adapted from Van Steen and Tanenbaum's Distributed Systems book

Last Time…

- Performance in Dist. Systems
  - Introduction
  - Performance metrics
  - Models

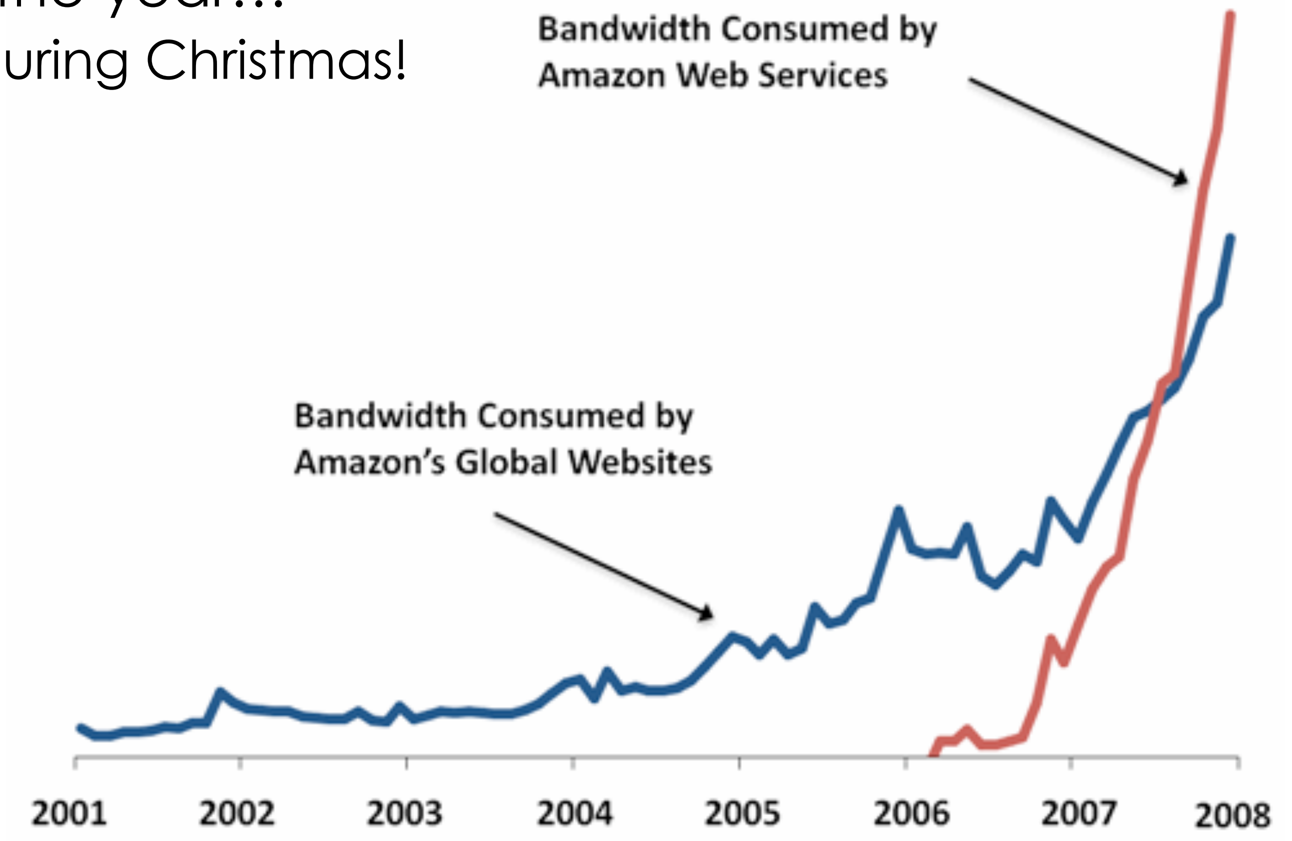
This Time…

- Cloud Application Architectures
  - IaaS vs PaaS
  - Multi-Tier
  - Microservices
  - Serverless
  - Big Data
  - Machine Learning

Amazon's Cloud

- Amazon built its cloud platform so that other people could pay for its infrastructure during the rest of the year…
  - Only needed peak capacity during Christmas!

- Now its cloud users are far bigger than its own sites



TYPES OF CLOUD SERVICES

- Infrastructure as a Service (**IaaS**)
  - Rent VMs, Containers, physical servers, disks, etc. by the hour
  - Examples: EC2, EBS, S3
  - Benefits?
  - Limitations?
- Other options: Function as a Service, Software as a Service, Resource as a Service…

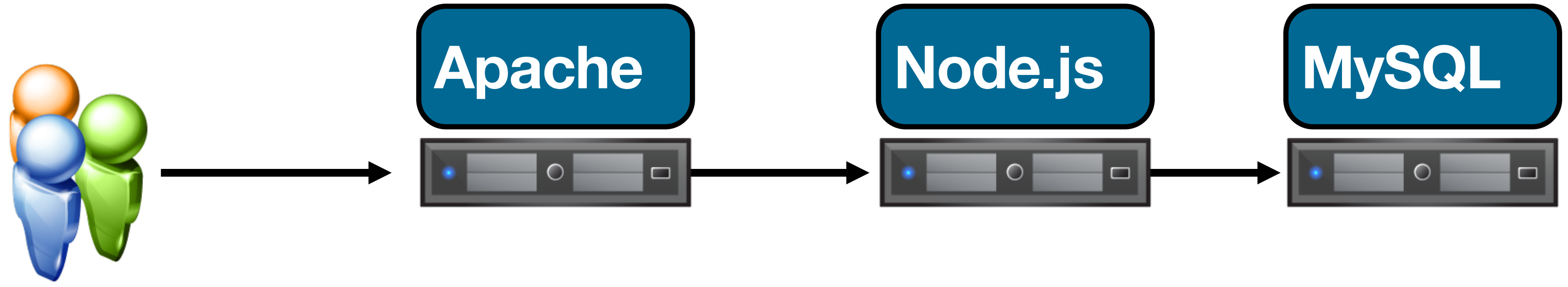
- Platform as a Service (**PaaS**)
  - Cloud provides a software layer on top of its resources
  - Exposes a programming API for users to develop cloud-based apps
  - Cloud provider manages all underlying resources (autoscaling)
  - Examples: Beanstalk, Lambda, EMR
  - Benefits?
  - Limitations?

Prof. Tim Wood & Prof. Roozbeh Haghnazar

Multi-Tier Applications

MULTI-TIER WEB APPLICATIONS

- Traditionally composed of 3 components
- Separation of duties:
  - **Front-end web server** for static content (Apache, lighttpd, nginx)
  - **Application (API) tier** for dynamic logic (PHP, Tomcat, node.js)
  - **Database back-end** holds state (MySQL, MongoDB, Postgres)
- Why divide up in this way?

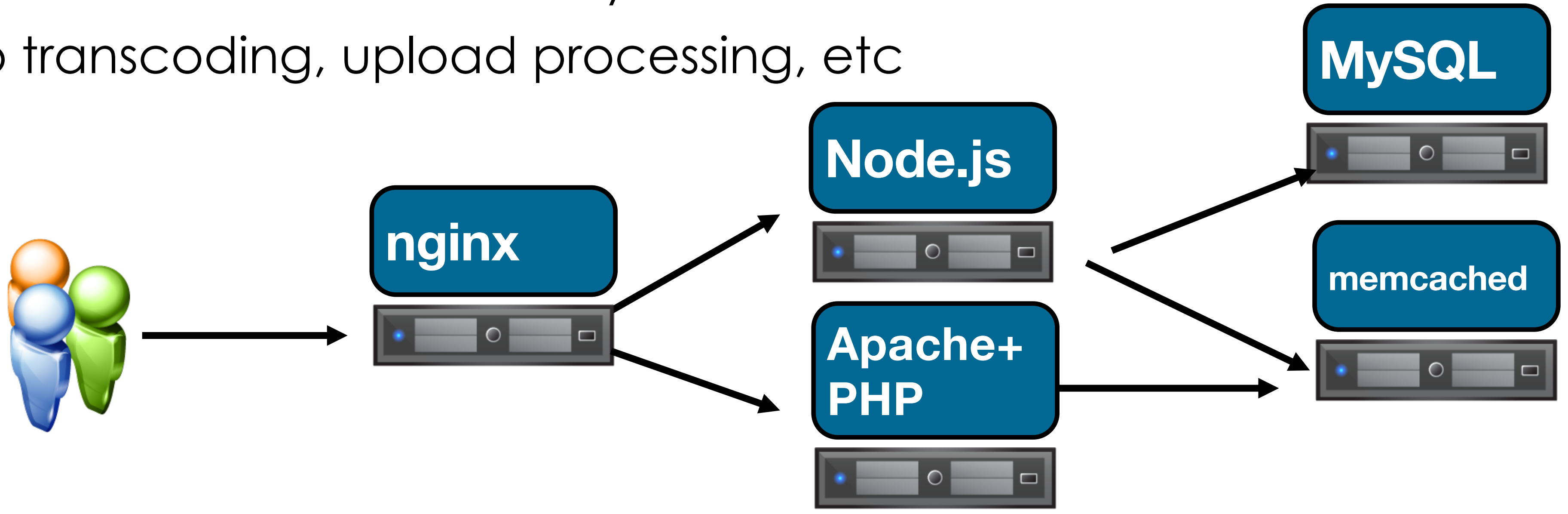


Stateful vs Stateless

- The multi-tier architecture is based largely around whether a tier needs to worry about state
- Front-end - totally **stateless**
  - There is no data that must be maintained by the server to handle subsequent requests
- Application tier - maintains **per-connection state**
  - There is some temporary data related to each user, e.g., my shopping cart
  - May not be critical for reliability - might just store in memory
- Database tier - global state
  - Maintains the global data that application tier might need
  - Persists state and ensures it is consistent

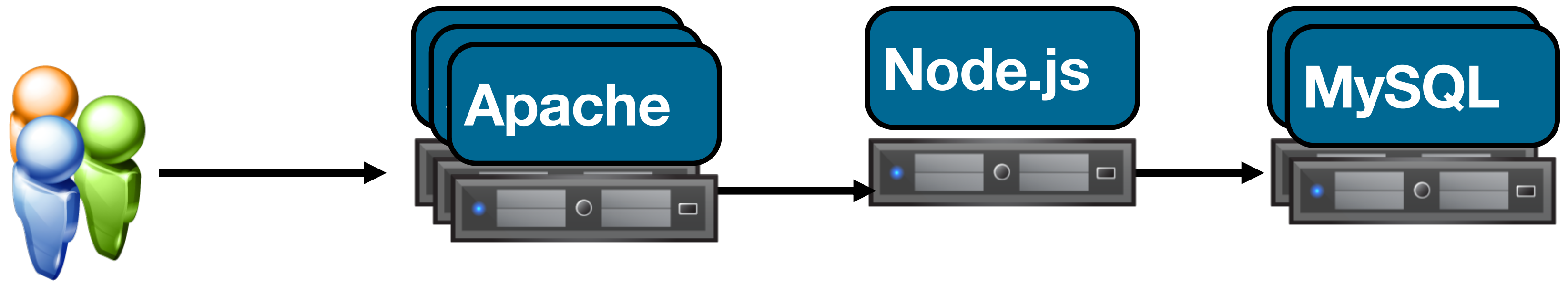
N-Tier Web Applications

- Sometimes 3 tiers isn't quite right
- Database is often a bottleneck
  - Add a cache! (stateful, but not persistent)
- Authentication or other security services could be another tier
- Video transcoding, upload processing, etc



Replicated N-Tier

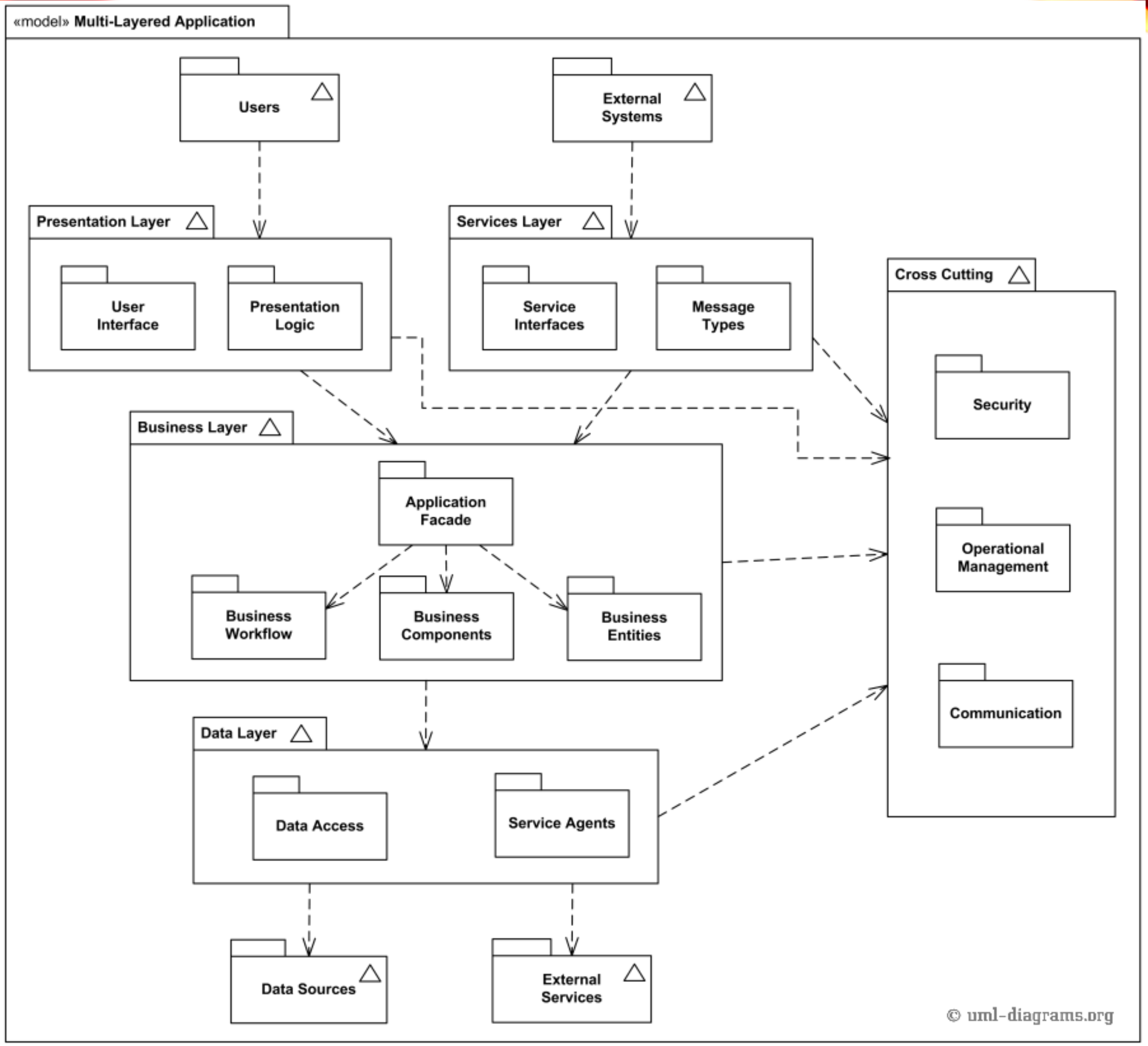
- Replicate the portions of the system that are likely to become overloaded
- How easy to scale…?
  - Apache serving static content
  - Tomcat Java application managing user shopping carts
  - MySQL cluster storing products and completed orders



Tune number of replicas based on demand at each tier

MULTI-LAYERED APPLICATION

An example of UML model diagram representing a model of a layered application, based on the Microsoft Application Achitecture Guide, 2nd Ed.

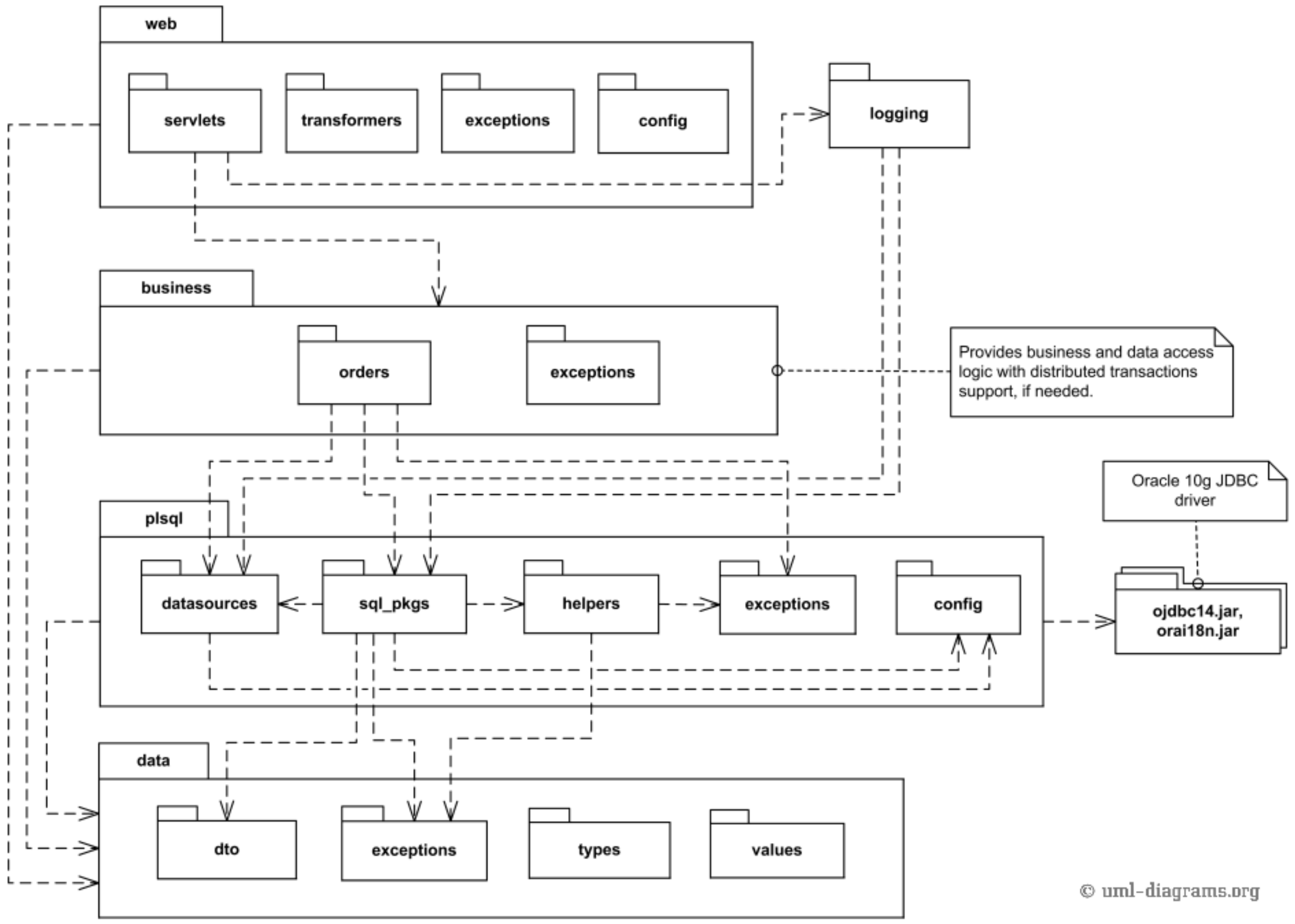


Multi-Layered Web Architecture UML Package Diagram Example

Dependencies between packages are created in such a way as to avoid circular dependencies between packages.

Higher level packages depend on lower level packages.

Packages belonging to the same level could depend on each other. Data transfer objects and common exceptions are used by packages at higher levels.



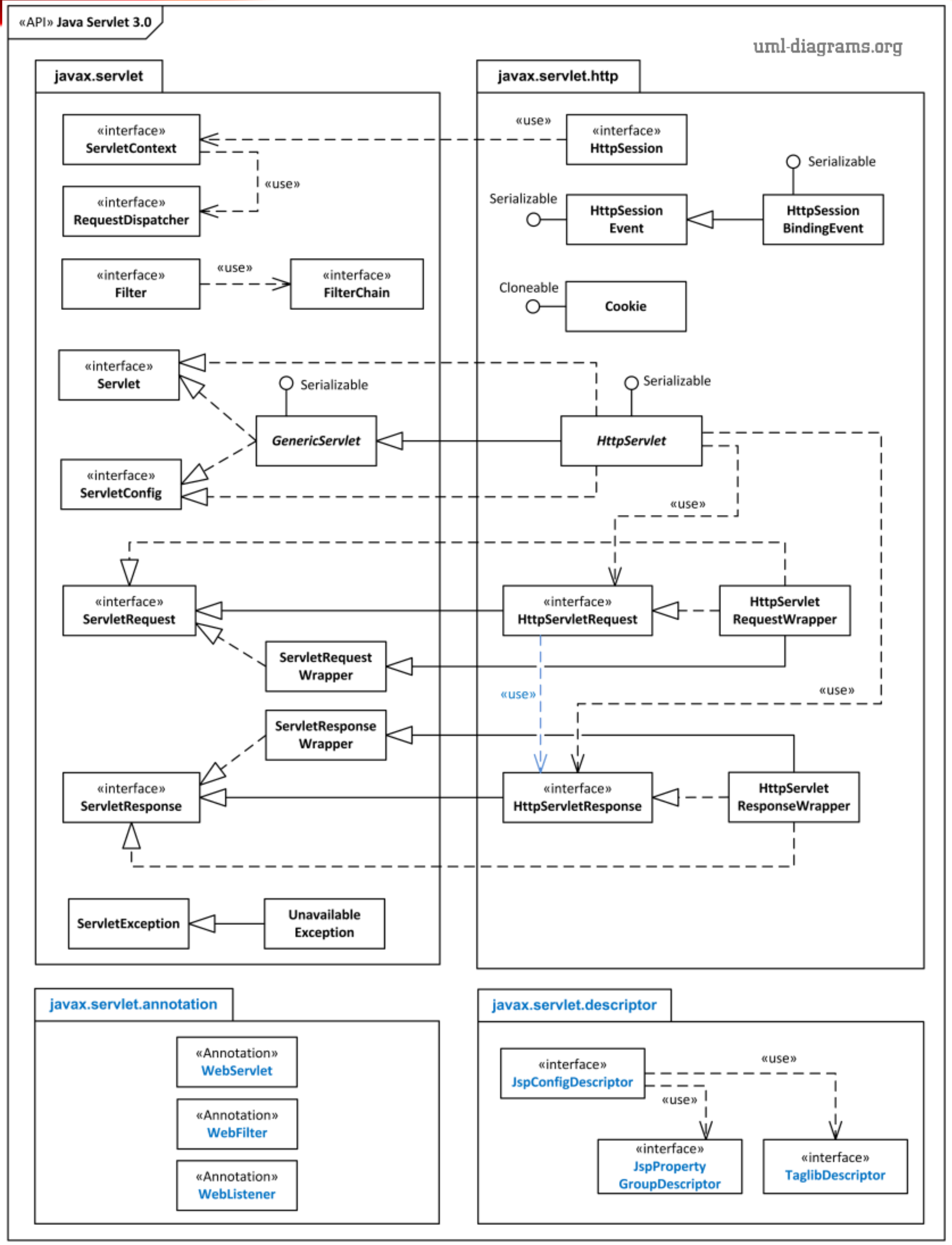
JAVA™ SERVLET 3.0 API

The first step can be Package Diagram

UML 2.5 doesn't have standard stereotype to support modeling of APIs

Java Servlet 3.0 API consists of four packages:

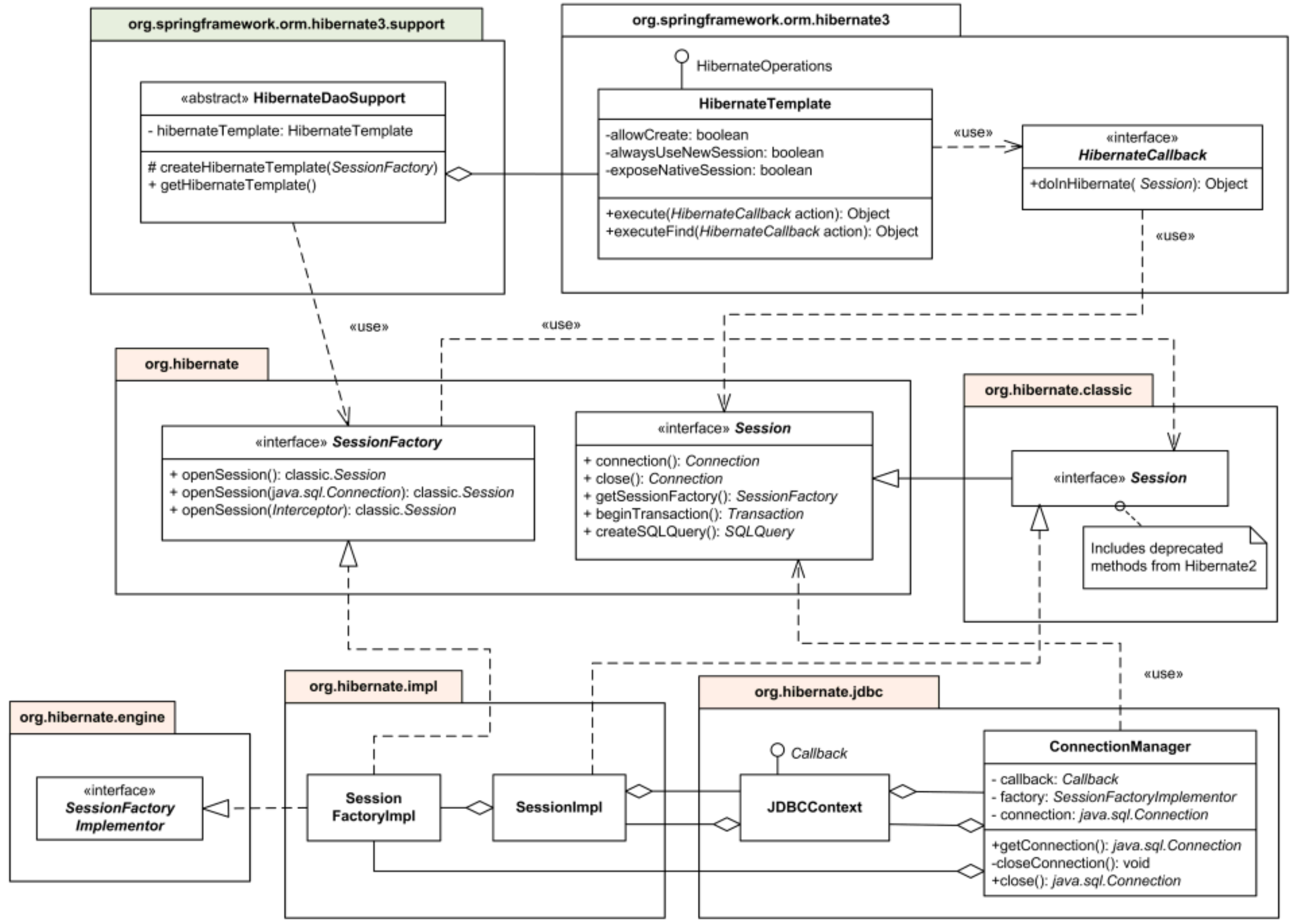
- **javax.servlet**,
- **javax.servlet.http**,
- **javax.servlet.annotation**, and
- **javax.servlet.descriptor**.



Spring and Hibernate Classes UML Package Diagram Example

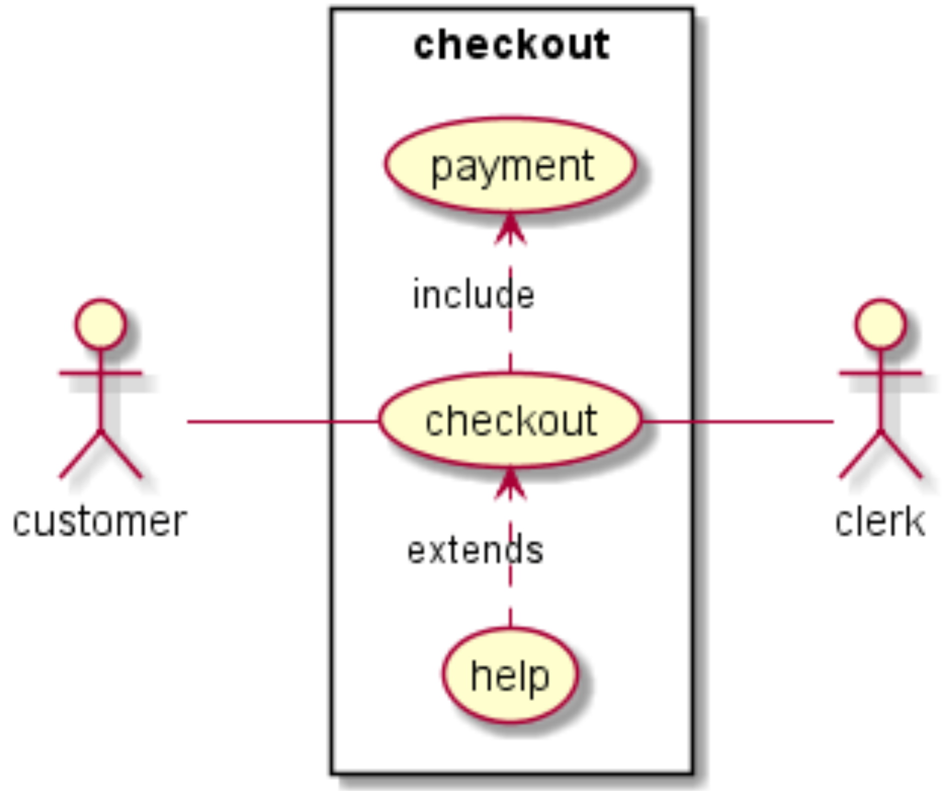
Support packages for object relational mappers comply with Spring's generic transaction and DAO exception hierarchies.

Spring framework hibernate3 packages use several Hibernate packages.


org.springframework.orm.hibernate3.support
«abstract» HibernateDaoSupport
- hibernateTemplate: HibernateTemplate
# createHibernateTemplate(SessionFactory)
+ getHibernateTemplate()
org.springframework.orm.hibernate3
HibernateOperations
HibernateTemplate
-allowCreate: boolean
-alwaysUseNewSession: boolean
-exposeNativeSession: boolean
+execute(HibernateCallback action): Object
+executeFind(HibernateCallback action): Object
«use»
«interface» HibernateCallback
+doInHibernate(Session): Object
«use»
«use»
«use»
org.hibernate
«interface» SessionFactory
+ openSession(): classic.Session
+ openSession(java.sql.Connection): classic.Session
+ openSession(Interceptor): classic.Session
«interface» Session
+ connection(): Connection
+ close(): Connection
+ getSessionFactory(): SessionFactory
+ beginTransaction(): Transaction
+ createSQLQuery(): SQLQuery
org.hibernate.classic
«interface» Session
Includes deprecated methods from Hibernate2
«use»
org.hibernate.engine
«interface» SessionFactory Implementor
org.hibernate.impl
Session FactoryImpl
SessionImpl
org.hibernate.jdbc
Callback
JDBCContext
ConnectionManager
- callback: Callback
- factory: SessionFactoryImplementor
- connection: java.sql.Connection
+getConnection(): java.sql.Connection
-closeConnection(): void
+close(): java.sql.Connection

PLANTUML

```
@startuml
left to right direction
skinparam packageStyle rectangle
actor customer
actor clerk
rectangle checkout {
  customer -- (checkout)
  (checkout) .> (payment) : include
  (help) .> (checkout) : extends
  (checkout) -- clerk
}
@enduml
```



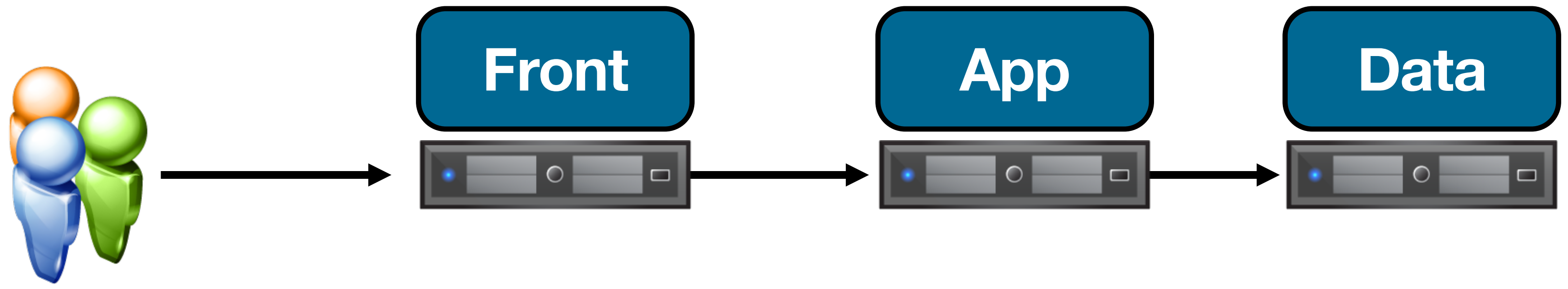
https://plantuml.com/use-case-diagram

Prof. Tim Wood & Prof. Roozbeh Haghnazar

Platform as a Service

What is PaaS?

- PaaS is a framework for developers to create fully-customized applications with limited management responsibilities in the cloud.
- Platform and environment to allow developers to build applications and services
- Simply accessible via their web browser



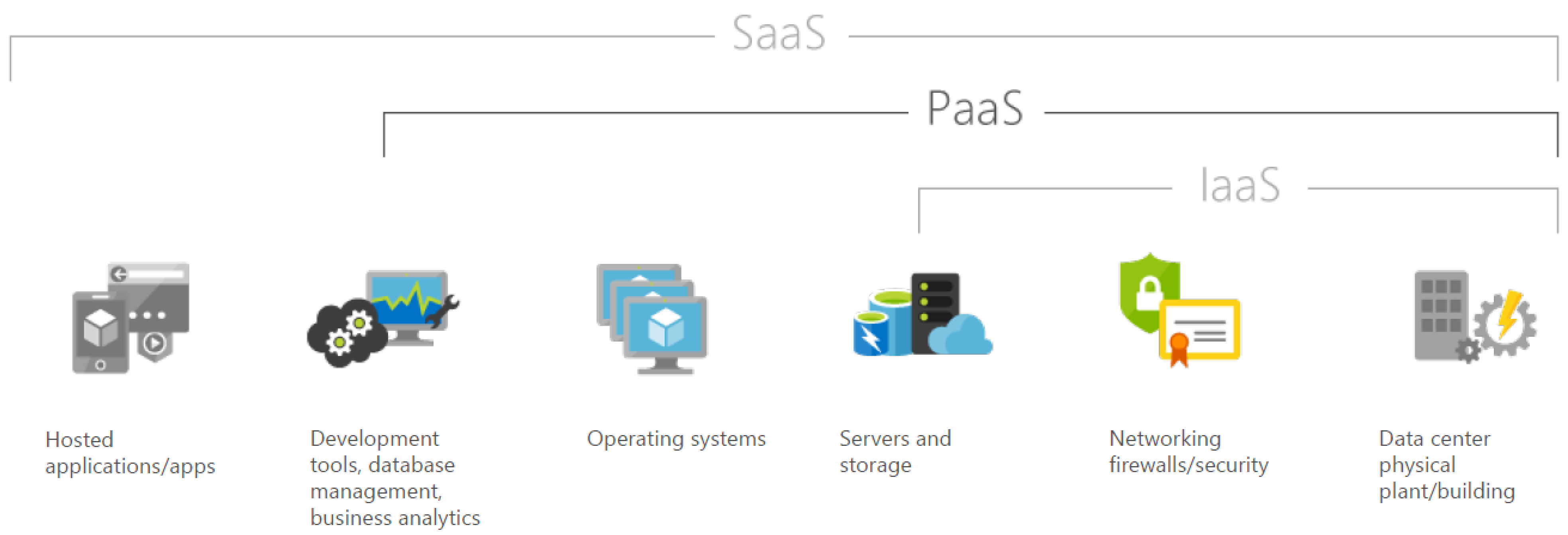
How does it work?

- This is important for enterprise businesses seeking agility as they implement a DevOps approach to software development.
- The bottom line is that Paas requires fewer management responsibilities, allowing for greater focus to be placed on development.
- Creation of Software Applications
- Pay-per-use
- Choice of features
- Management and support
- Automatic upgrades

Benefits to Application Developers

- No investment in physical infrastructure
- Make development possible for ‘non-experts’
- Flexibility & Adaptability
- Teams in various location can work together
- Security

PaaS in Azure


SaaS
PaaS
IaaS
Hosted applications/apps
Development tools, database management, business analytics
Operating systems
Servers and storage
Networking firewalls/security
Data center physical plant/building

PaaS in Azure

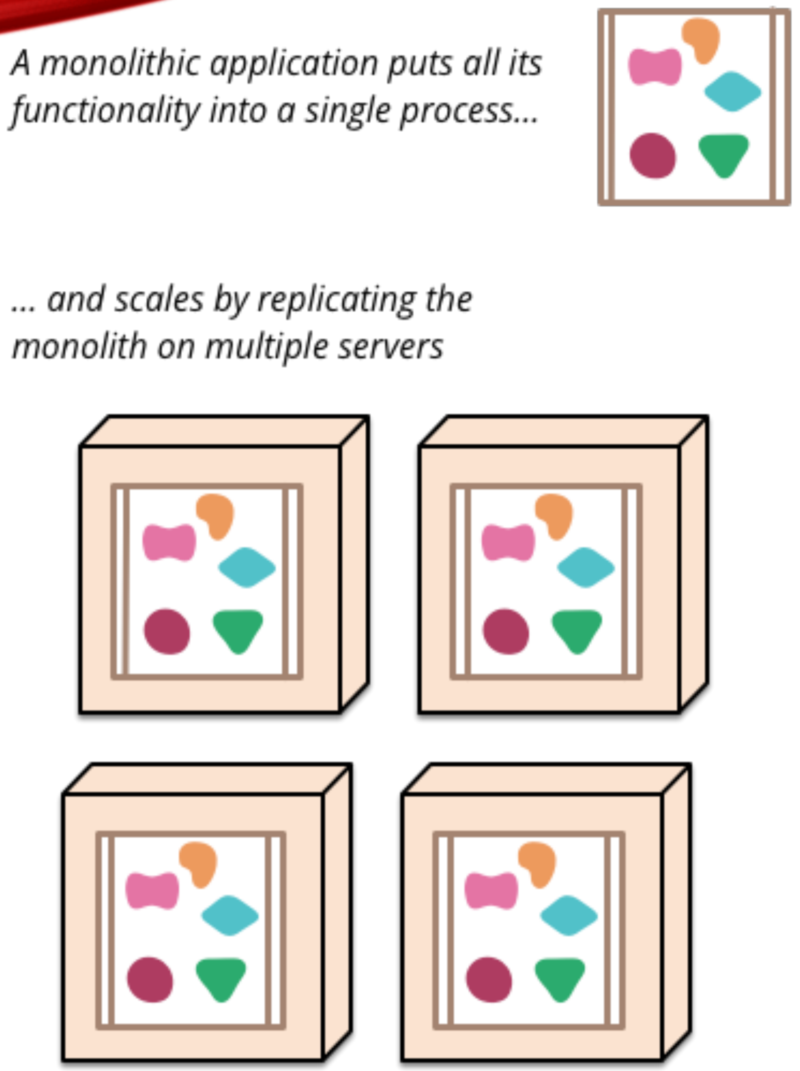
- Web Apps
- Mobile Apps
- Logic Apps
- Functions
- Web Jobs

Prof. Tim Wood & Prof. Roozbeh Haghnazar

Application Tier

Monolithic vs Microservices

APPLICATION TIER



A monolithic application puts all its functionality into a single process...

... and scales by replicating the monolith on multiple servers

http://martinfowler.com/articles/microservices.html

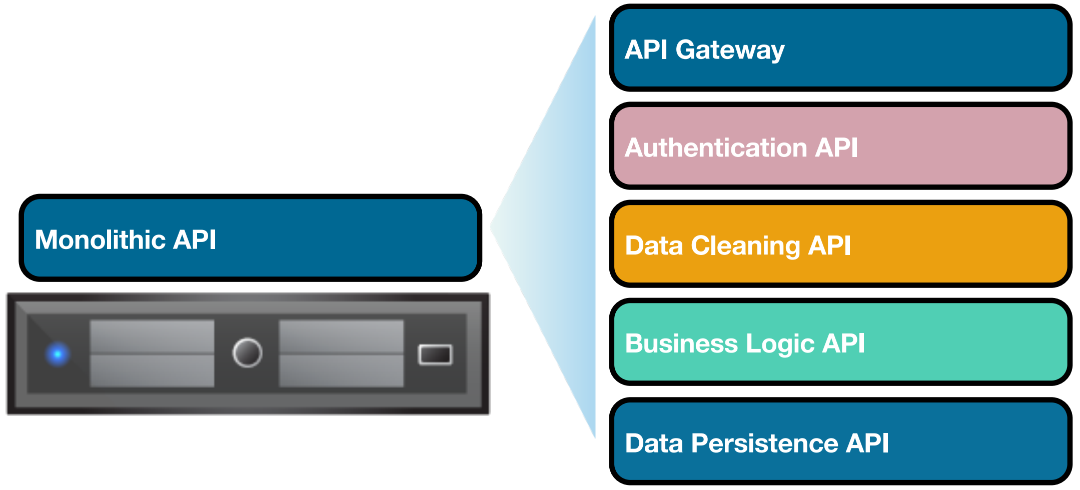
Problems with Monolithic approach?

Monolithic Challenges

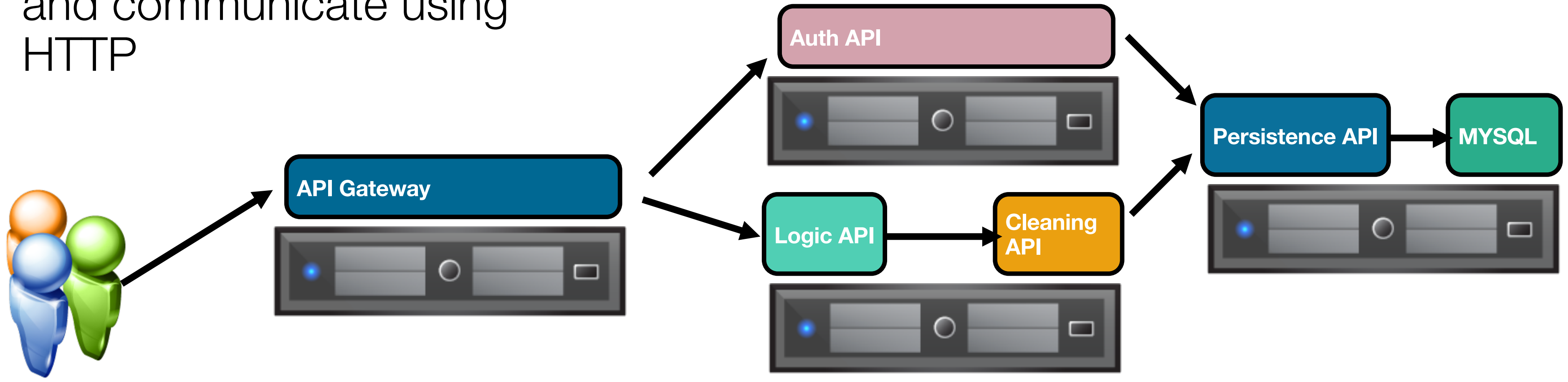
- **Scalability:** Need to possibly use both up and out to reach performance goals. Hard to scale things like databases.
- **Reliability:** A fault/memory leak in a single place can crash the entire app.
- **Orchestration:** You need to rebuild and deploy the entire application every time you make a change.
- **Code Complexity:** Code turns into spaghetti due to too many things happening at once. Hard to refactor features.
- **Upgradeability**: Moving to newer tech stacks requires converting the entire app at once.

MICROSERVICES

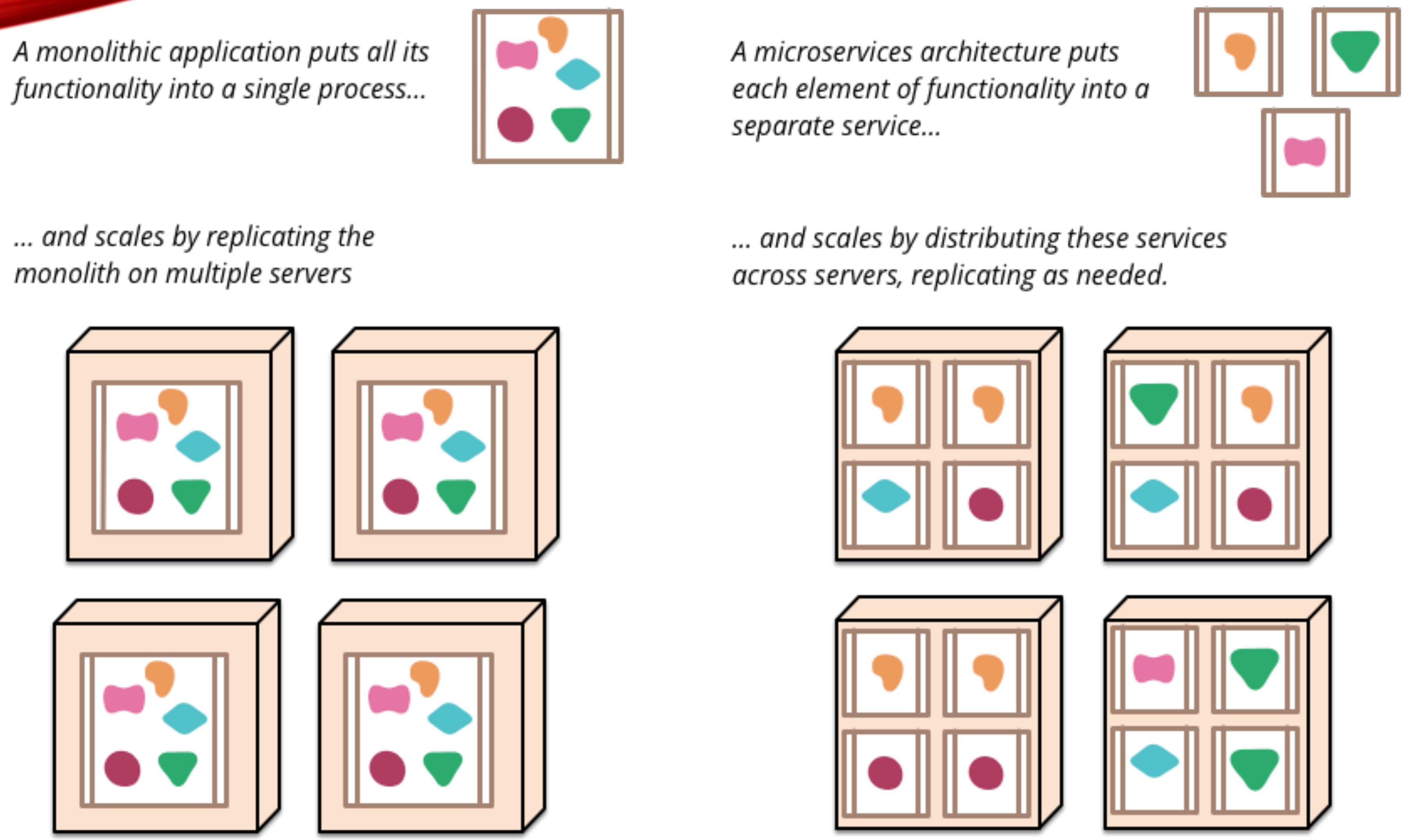
- Take your application API and split it into smaller components based on function.



API's are then distributed across many machines and communicate using HTTP



MICROSERVICES



A monolithic application puts all its functionality into a single process...

... and scales by replicating the monolith on multiple servers

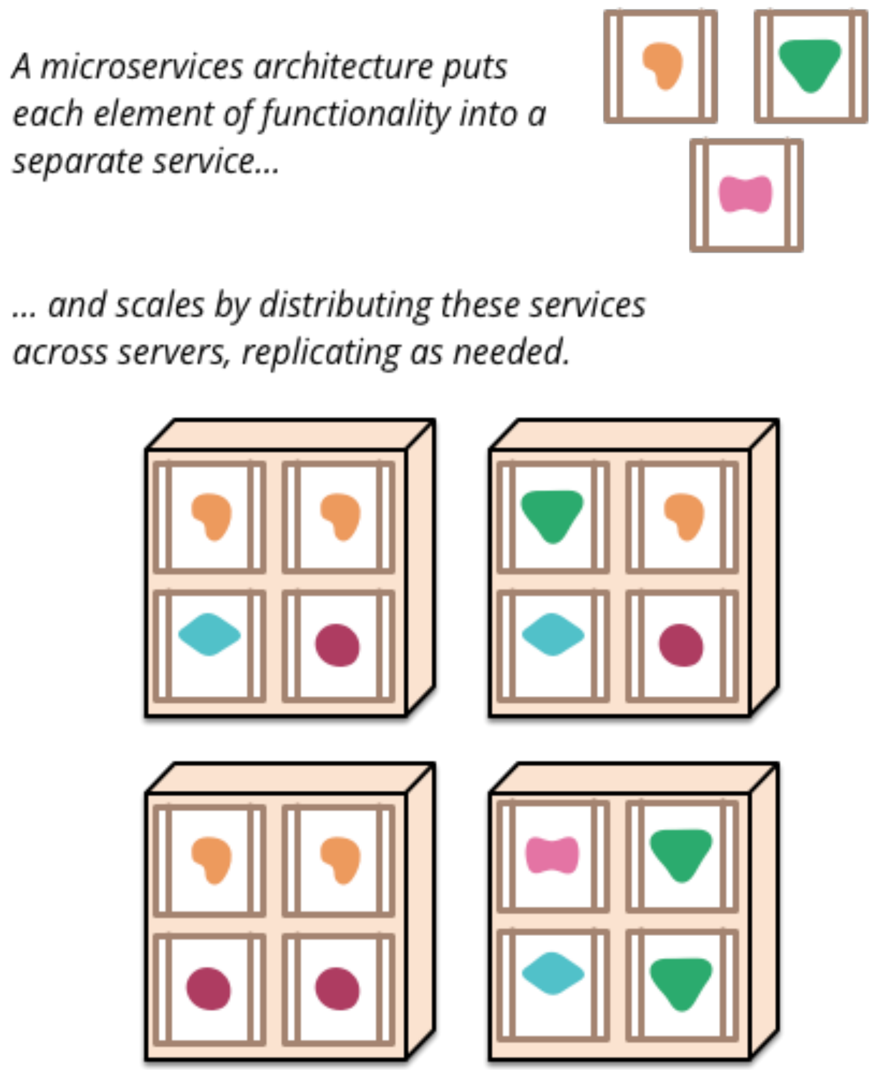
A microservices architecture puts each element of functionality into a separate service...

... and scales by distributing these services across servers, replicating as needed.

Read more: https://martinfowler.com/articles/microservices.html

MICROSERVICES

Challenges with Microservices approach?


A microservices architecture puts each element of functionality into a separate service...
... and scales by distributing these services across servers, replicating as needed.

Read more: https://martinfowler.com/articles/microservices.html

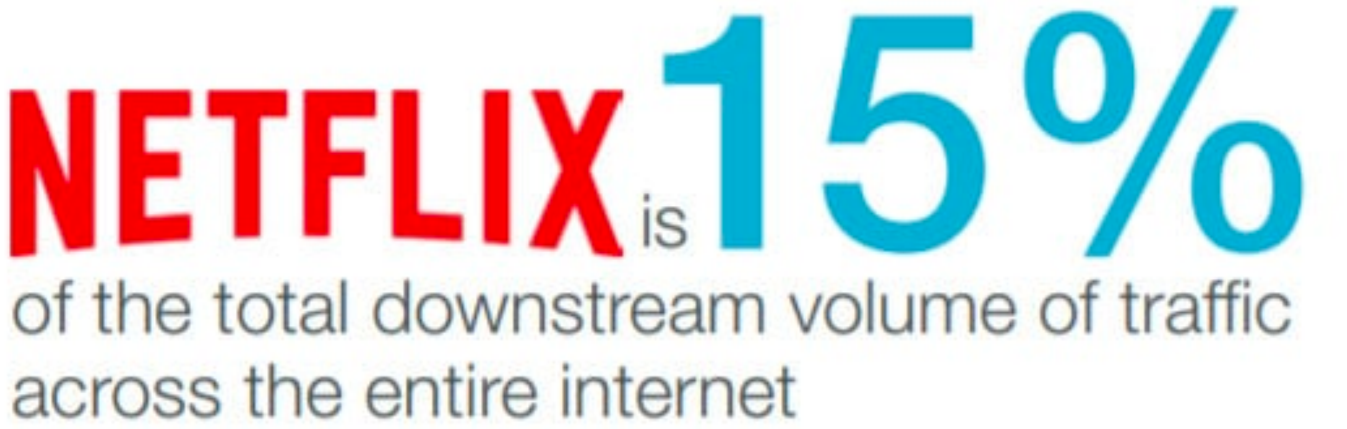
MICROSERVICES CHALLENGES

- **Discovery**: how to find a service you want?
- **Scalability**: how to replicate services for speed?
- **Openness**: how to agree on a message protocol?
- **Fault tolerance**: how to handle failed services?

All distributed systems face these challenges, microservices just increases the scale and diversity…

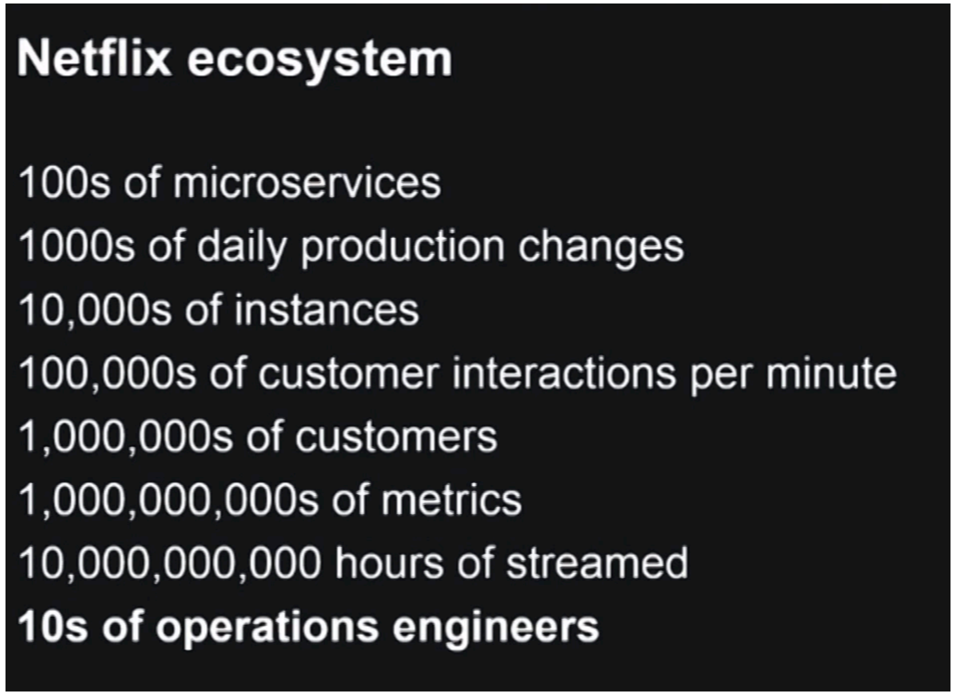
NETFLIX

- 20th most popular website according to Alexa
- Zero of their own servers
  - All infrastructure is on AWS (2016-2018)
  - Recently starting to build out their own Content Delivery Network


NETFLIX is 15% of the total downstream volume of traffic across the entire internet

NETFLIX

- One of the first to really push microservices
  - Known for their DevOps
  - Fast paced, frequent updates, must always be available
- 700+ microservices
- Deployed across 10,000s of VMs and containers



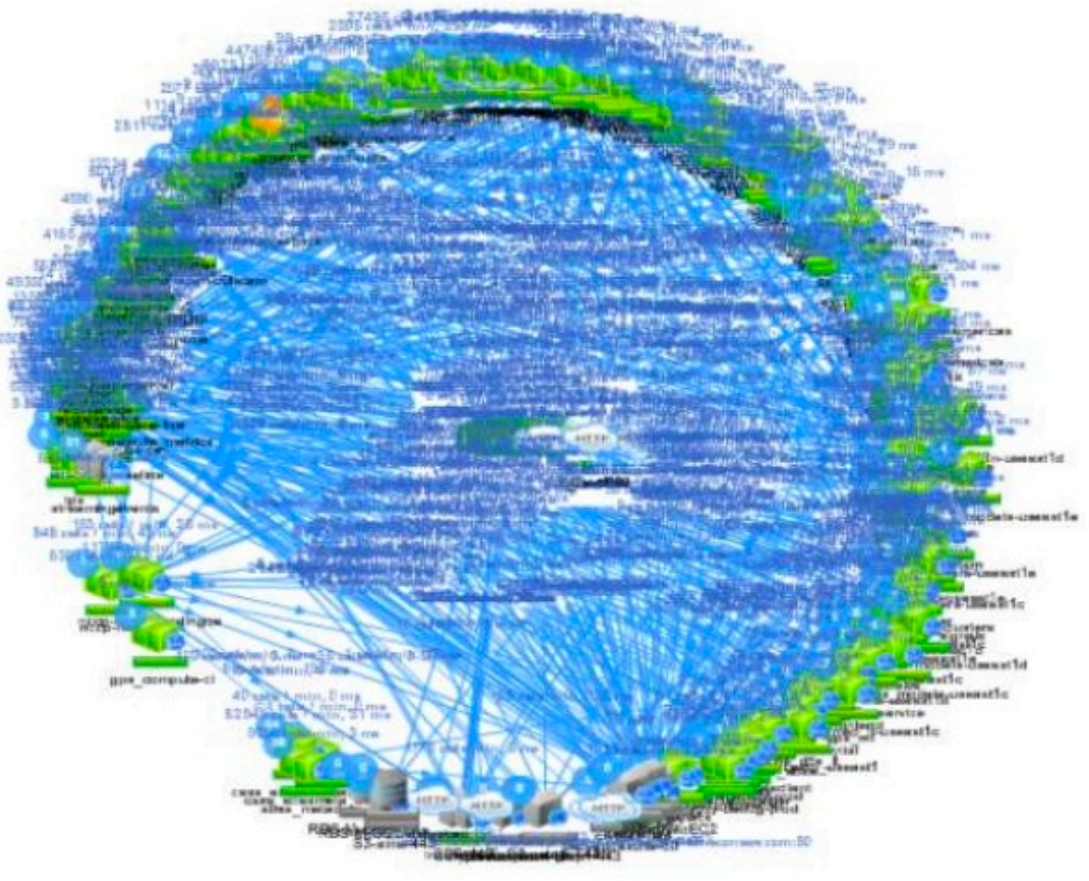
Netflix ecosystem

100s of microservices
1000s of daily production changes
10,000s of instances
100,000s of customer interactions per minute
1,000,000s of customers
1,000,000,000s of metrics
10,000,000,000 hours of streamed
10s of operations engineers

Netflix tech talk: https://www.youtube.com/watch?v=CZ3wIuvmHeM

NETFLIX "DEATHSTAR"

- Microservice architecture results in a extremely distributed application
  - Can be very difficult to manage and understand how it is working at scale
- What if there are failures?
- How to know if everything is working correctly?



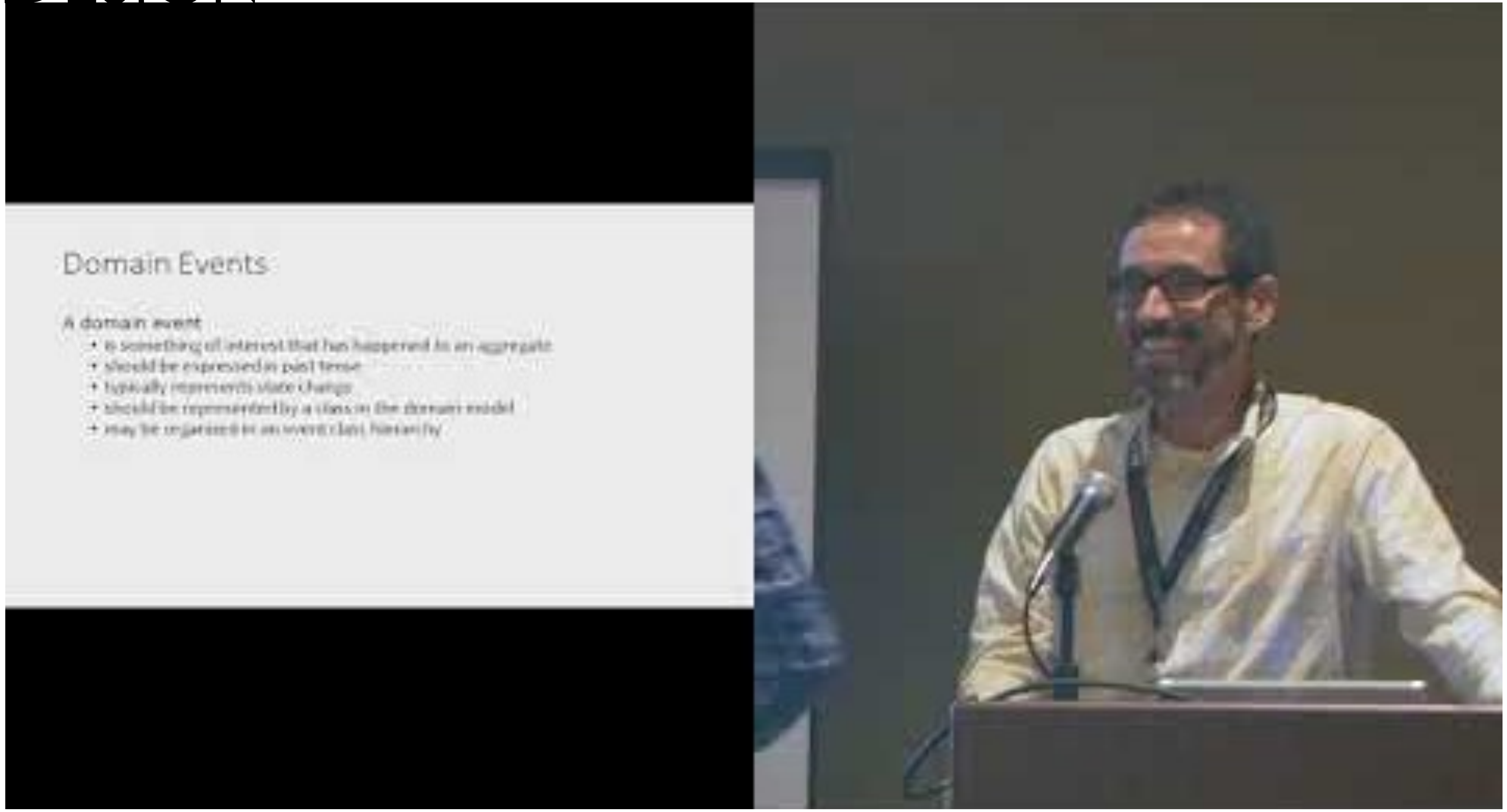
NETFLIX CHAOS MONKEY

- Idea: If my system can handle failures, then I don't need to know exactly how all the pieces themselves interact!
- Chaos Monkey:
  - Randomly terminate VMs and containers in the production environment
  - Ensure that the overall system keeps operating
  - Run this 24/7

http://principlesofchaos.org/

Make failures the common case, not an unknown!

Microservice Architecture Modeling with Domain Driven Design



Domain Events

A domain event

- is something of interest that has happened to an aggregate
- should be expressed in past tense
- typically represents state change
- should be represented by a class in the domain model
- may be organized in an event class hierarchy

https://www.umlzone.com/videos/modeling-microservices-with-domain-driven-design/

Prof. Tim Wood & Prof. Roozbeh Haghnazar

Serverless Computing

Serverless Computing

- Trendy architecture that improves the agility of microservices
- What does "serverless" mean?

APACHE OpenWhisk™

AWS Lambda

Azure Functions

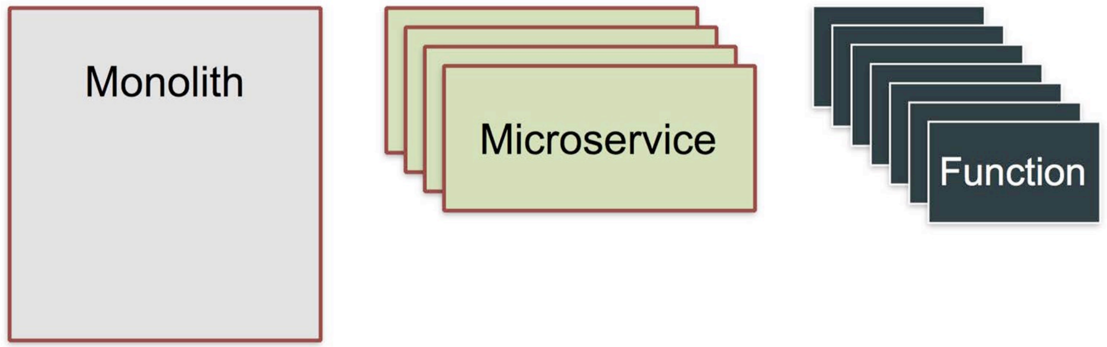
Google Cloud Functions

Serverless Computing (FaaS)

- Trendy architecture that improves the agility of microservices
- What does “serverless” mean?
- You still need a server!
- BUT, your services will not always be running

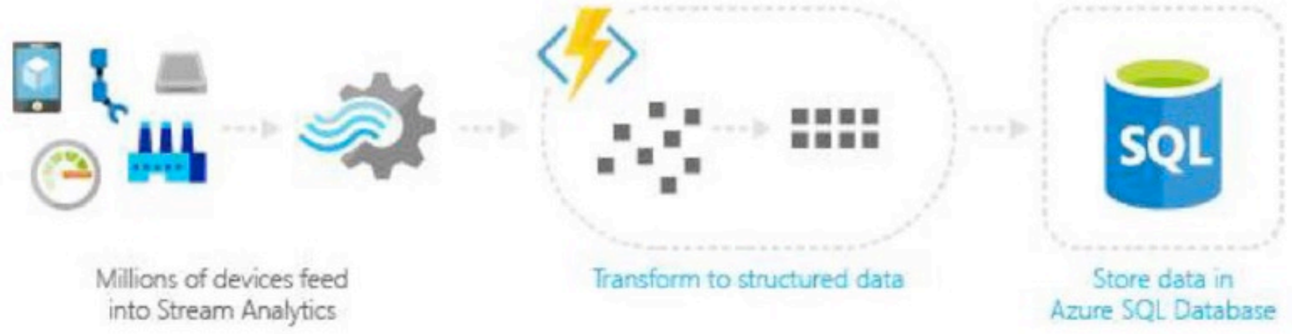
- Key idea: only **instantiate** a service when a user makes a request for that functionality; **charge** the cloud customer based on number of requests

- How will this work for stateful vs stateless services?



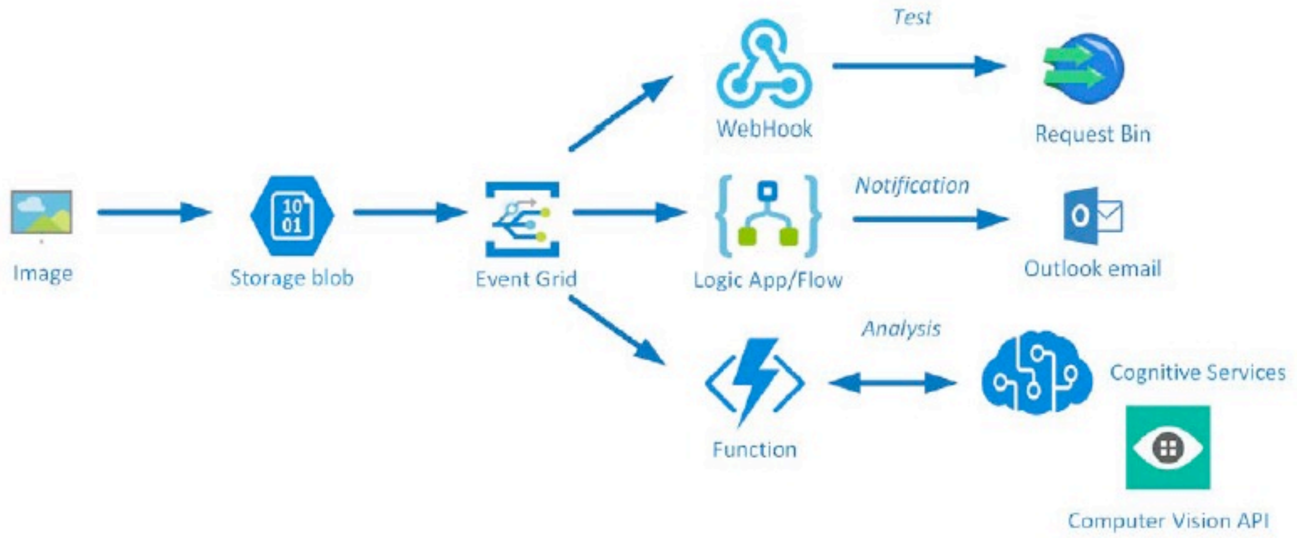
Usage examples of Serverless

- IoT (Weather stations)



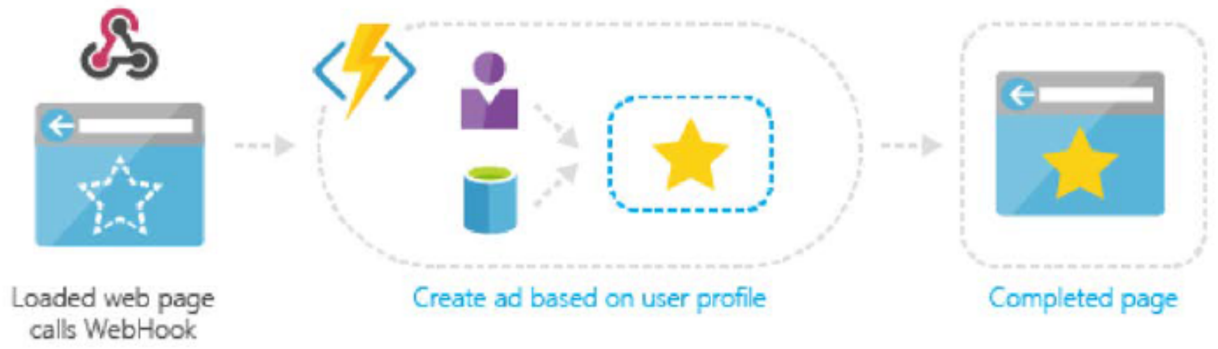
Millions of devices feed into Stream Analytics
Transform to structured data
SQL
Store data in Azure SQL Database

- Data processing (Image Manipulation)

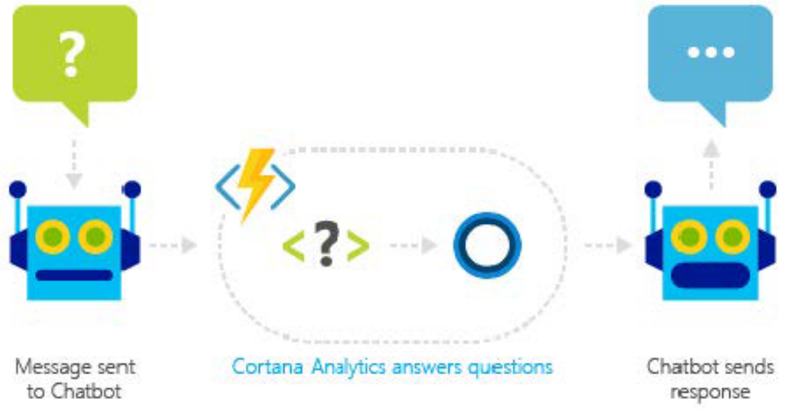


Usage examples of Serverless

- Web Application (Weather stations)


Loaded web page calls WebHook
Create ad based on user profile
Completed page

- Chatbots (Event-Driven Architecture for chatbot)



Serverless in Azure

Storage
Functions
Logic App
Media Services
Cortana Management Suite
App Service
Document DB
CDN
Machine Learning
OMS Management Suit
Visual Studio Services
Traffic Manager
Active Directory
Key Vault
App Insights
Cognitive Services
Embedded Power BI
Search
IoT Hub
Service Bus
Notification Hub
Stream Analytics
Hockey App
Scheduler

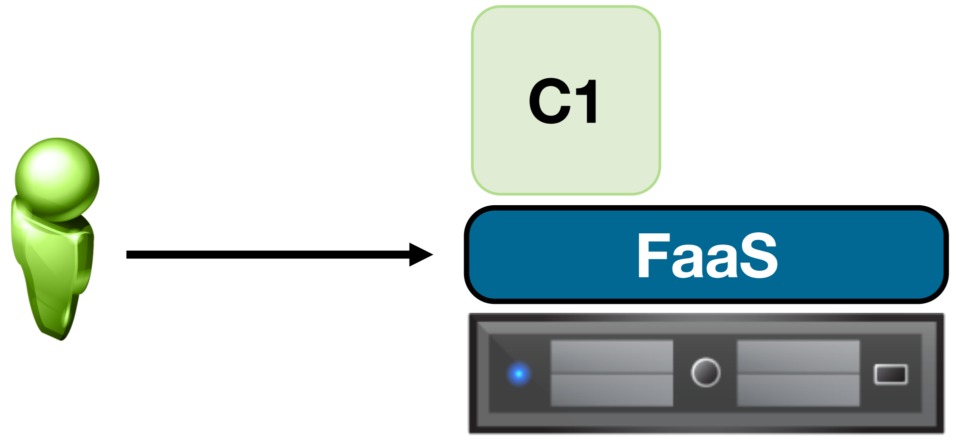
SERVERLESS STARTUP

- Function as a Service platfoms (Lambda, GCF, Azure Functions, etc)
  - Define a stateless "function" to execute for each request
  - A container will be instantiated to handle the first request
  - The same container will be used until it times out or is killed

No workload means no resources being used!

SERVERLESS STARTUP

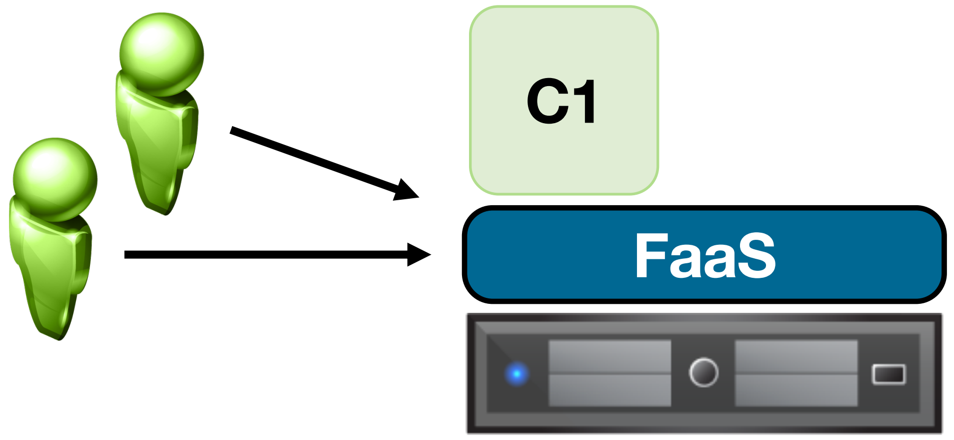
- Function as a Service platfoms (Lambda, GCF, Azure Functions, etc)
  - Define a stateless "function" to execute for each request
  - A container will be instantiated to handle the first request
  - The same container will be used until it times out or is killed



Request arrives, start green container

SERVERLESS STARTUP

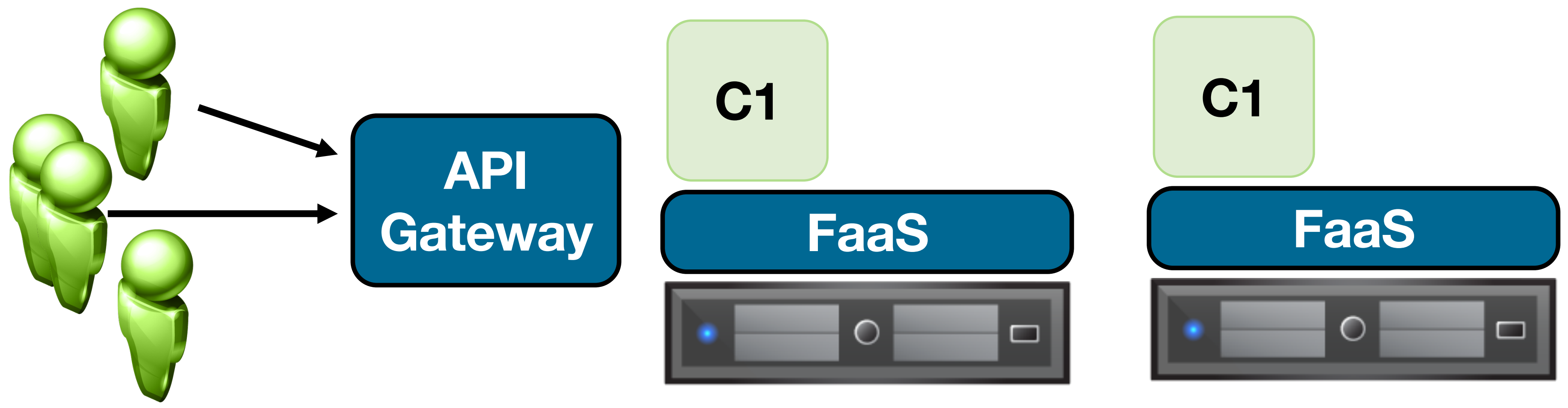
- Function as a Service platfoms (Lambda, GCF, Azure Functions, etc)
  - Define a stateless "function" to execute for each request
  - A container will be instantiated to handle the first request
  - The same container will be used until it times out or is killed



Reuse that container for subsequent requests

Serverless Startup

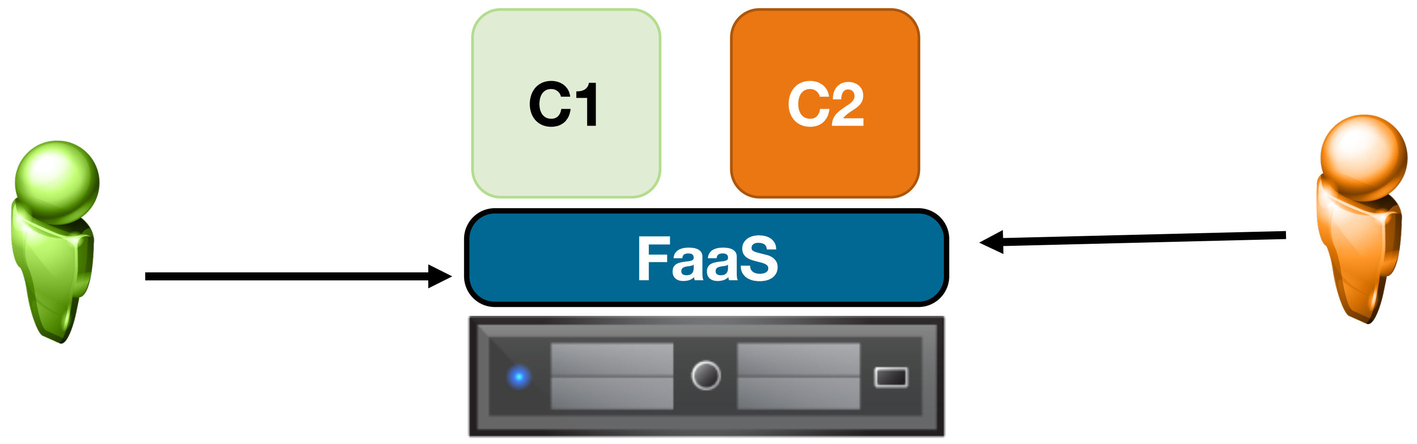
- Function as a Service platfoms (Lambda, GCF, Azure Functions, etc)
  - Define a stateless "function" to execute for each request
  - A container will be instantiated to handle the first request
  - The same container will be used until it times out or is killed



Add more replicas if workload exceeds capacity

Serverless Startup

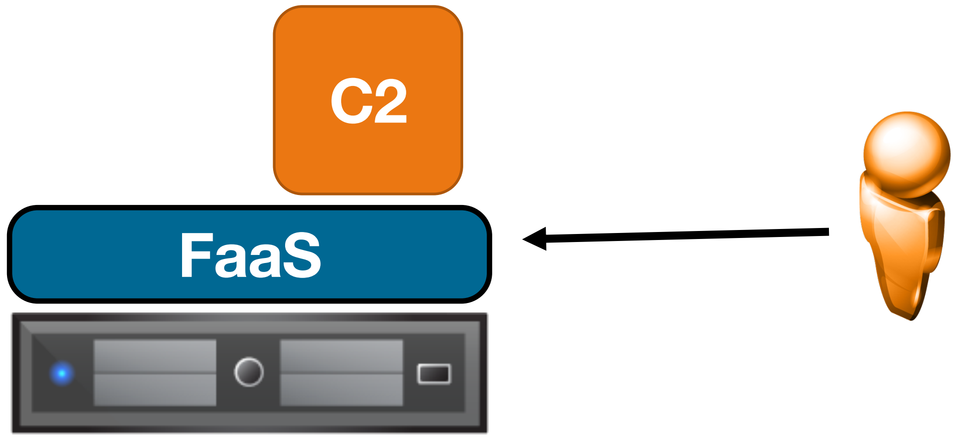
- Function as a Service platfoms (Lambda, GCF, Azure Functions, etc)
  - Define a stateless "function" to execute for each request
  - A container will be instantiated to handle the first request
  - The same container will be used until it times out or is killed



Start new container if user needs a different function

Serverless Startup

- Function as a Service platfoms (Lambda, GCF, Azure Functions, etc)
  - Define a stateless "function" to execute for each request
  - A container will be instantiated to handle the first request
  - The same container will be used until it times out or is killed

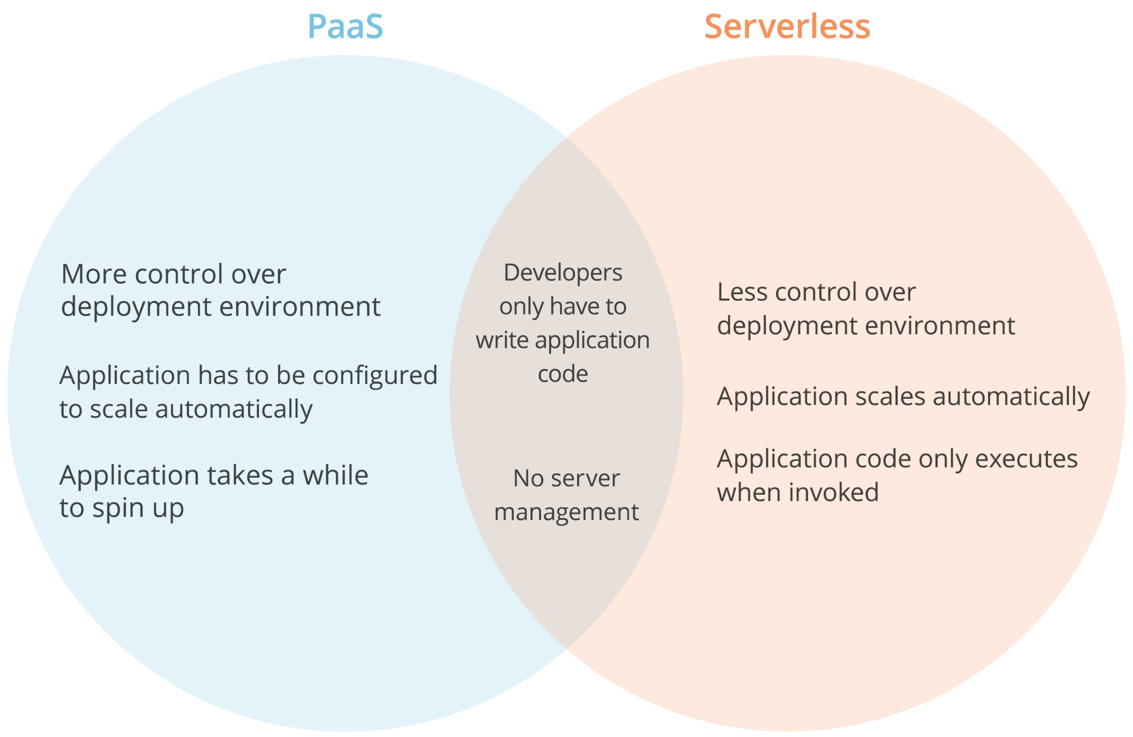


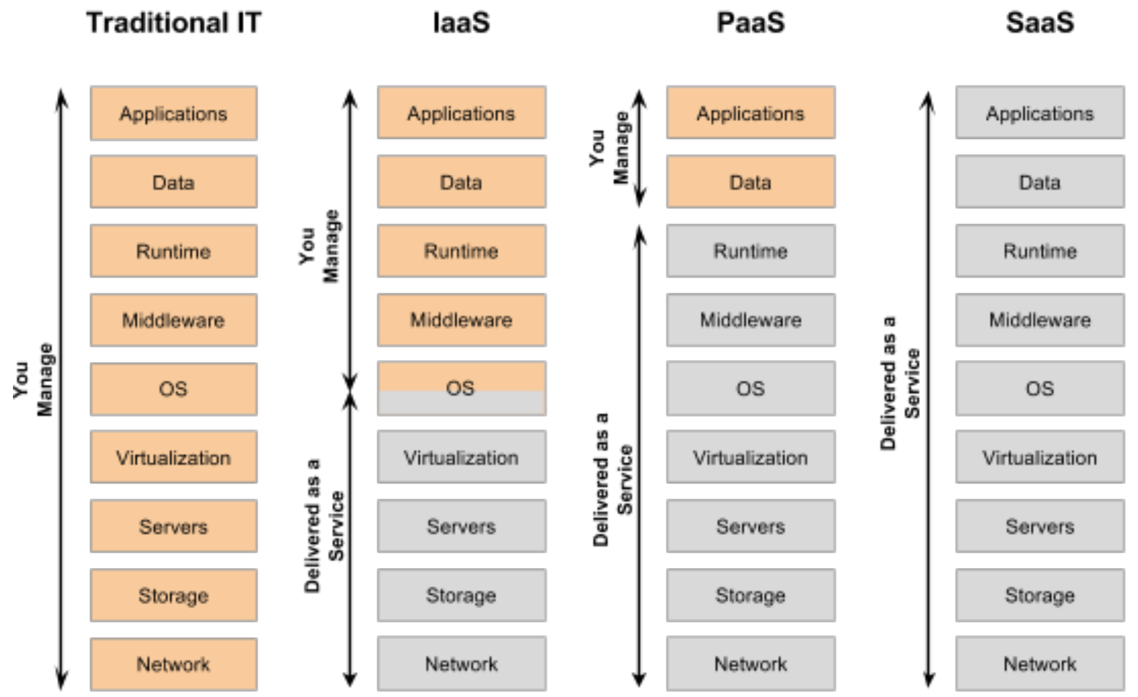
Stop old containers once not in use

Serverless Pros/Cons

- Benefits:
  - Simple for developer when auto scaling up
  - Pay for exactly what we use (at second granularity)
  - Efficient use of resources (auto scale up and down based on requests)
  - don’t worry about reliability/server management at all
- Drawbacks:
  - Limited functionality (stateless, limited programming model)
  - High latency for first request to each container
  - Some container layer overheads plus the lambda gateway and routing overheads
  - Potentially higher and unpredictable costs
  - Difficult to debug / monitor behavior
  - Security

Serverless vs PaaS?


PaaS
Serverless
More control over deployment environment
Application has to be configured to scale automatically
Application takes a while to spin up
Developers only have to write application code
No server management
Less control over deployment environment
Application scales automatically
Application code only executes when invoked



Prof. Tim Wood & Prof. Roozbeh Haghnazar

Serverless Demo