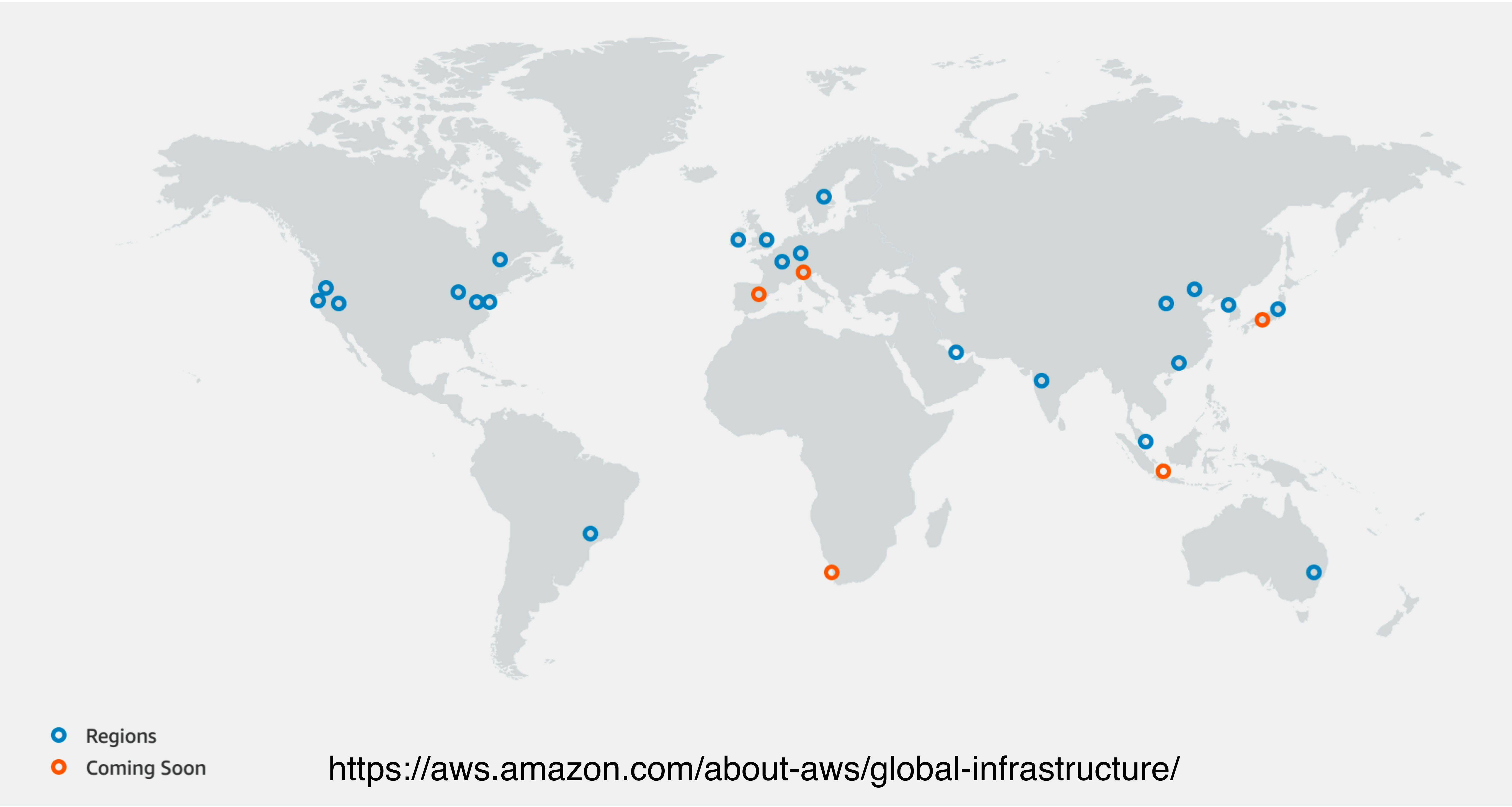
Distributed Clouds

GW Advanced Networking and Distributed Systems

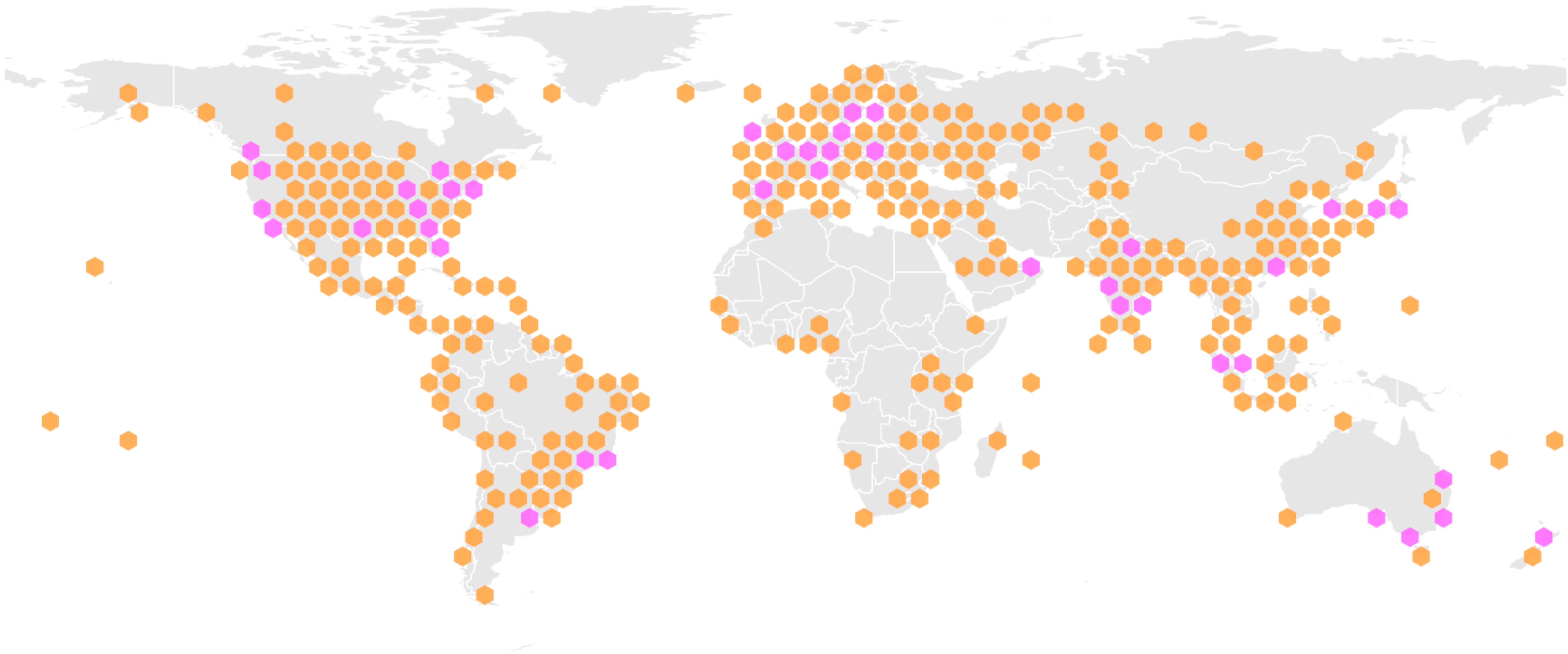
Tim Wood and Lucas Chaufournier

Amazon's Cloud



https://aws.amazon.com/about-aws/global-infrastructure/

Akamai CDN



https://www.akamai.com/us/en/resources/visualizing-akamai/media-delivery-map.jsp

Content Delivery Networks

CDNs are a form of distributed cloud

Lots of "points of presence" (PoPs)

- Each with small number of servers (6-6000)

Cache popular content close to users

- Why?

Caching

What kind of cloud applications can we cache?

What can make caching difficult?

CDN Demo

Thanks Ben and Ethan!

Let's compare..

- A cat stored in S3 (amazon's storage)
  - http://bendogpicture.s3-website-ap-southeast-1.amazonaws.com/
- A cat stored in Amazon Cloudfront
  - http://d14mfeaqszawbm.cloudfront.net/

https://tools.keycdn.com/performance

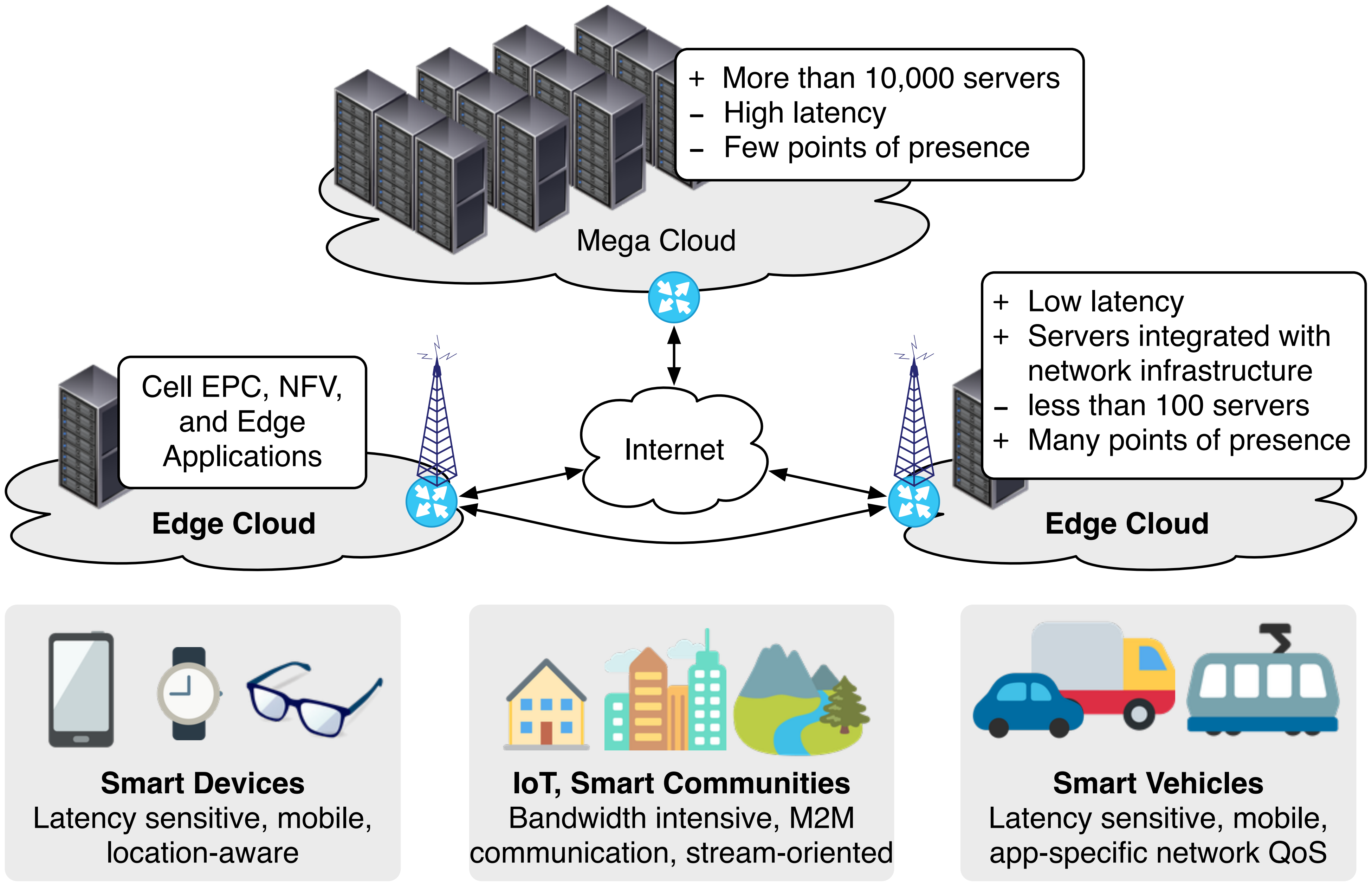
- Tests web request speed from 14 locations around the world

What do you expect to see?

Cloudfront Demo

Edge Computing: The Future?

Can we make CDNs more dynamic and programmable?



Mega Cloud
+ More than 10,000 servers
– High latency
– Few points of presence
Internet
Cell EPC, NFV, and Edge Applications
Edge Cloud
+ Low latency
+ Servers integrated with network infrastructure
– less than 100 servers
+ Many points of presence
Edge Cloud
Smart Devices
Latency sensitive, mobile, location-aware
IoT, Smart Communities
Bandwidth intensive, M2M communication, stream-oriented
Smart Vehicles
Latency sensitive, mobile, app-specific network QoS

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

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

How will this work for stateful vs stateless services?

Serverless Startup

AWS Lambda

- 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

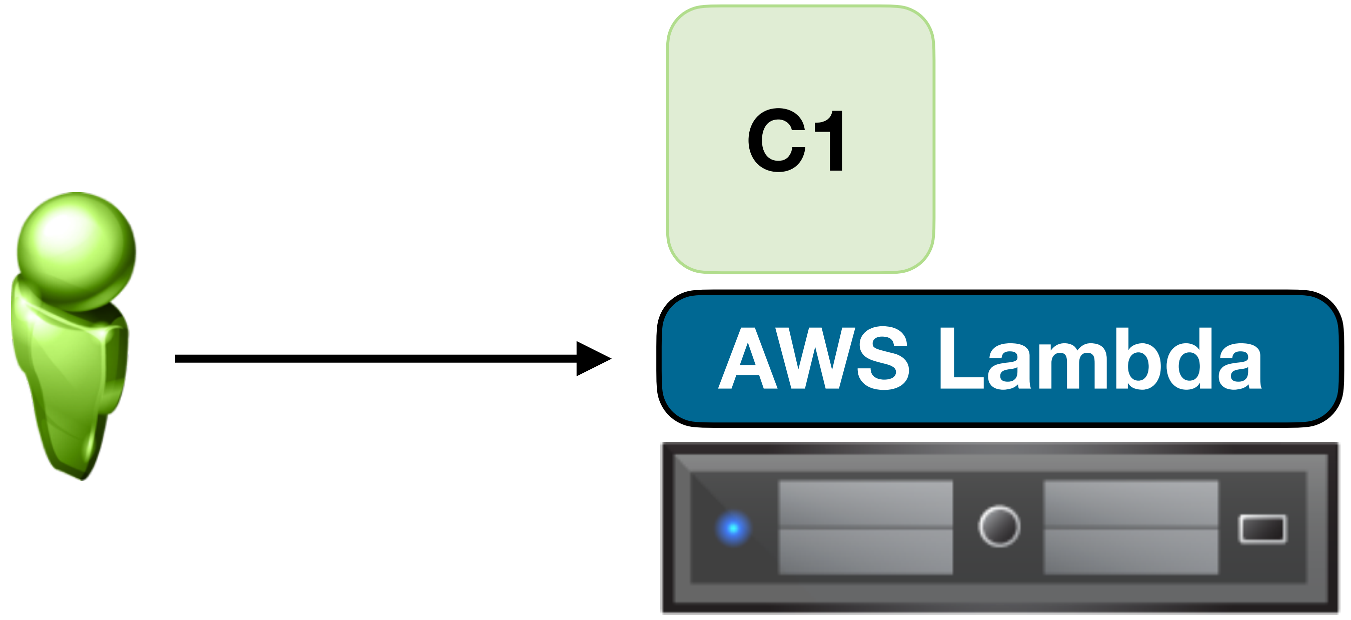
No workload means no resources being used!

Serverless Startup

AWS Lambda

- 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

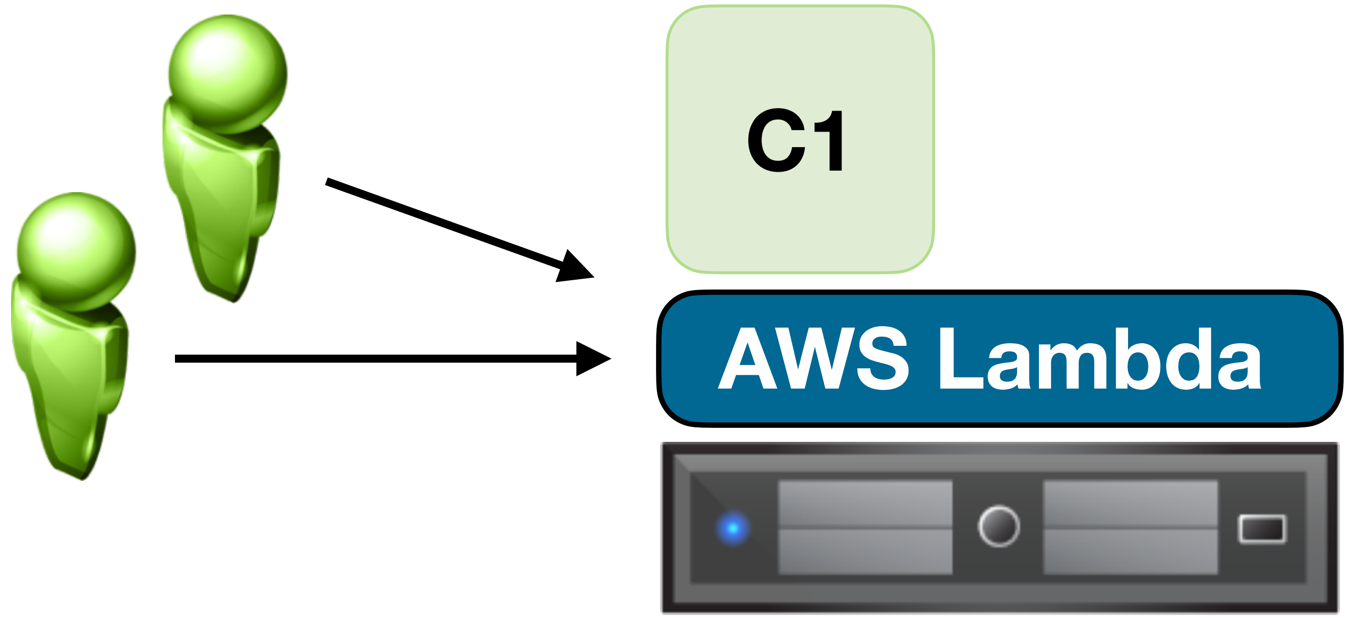


Serverless Startup

AWS Lambda

- 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

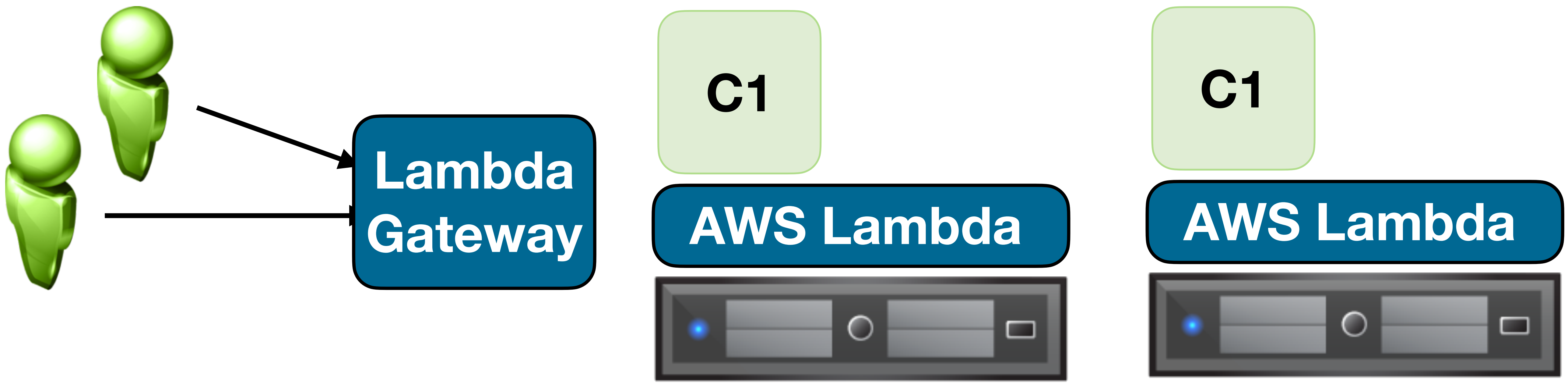


Serverless Startup

AWS Lambda

- 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

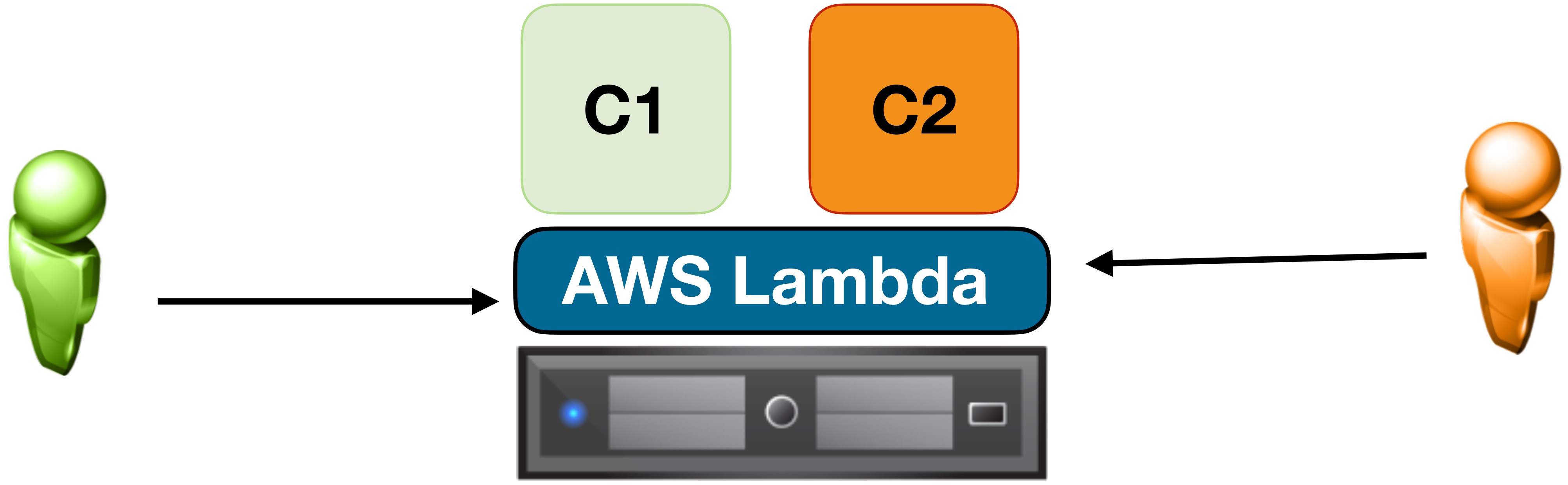


Serverless Startup

AWS Lambda

- 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

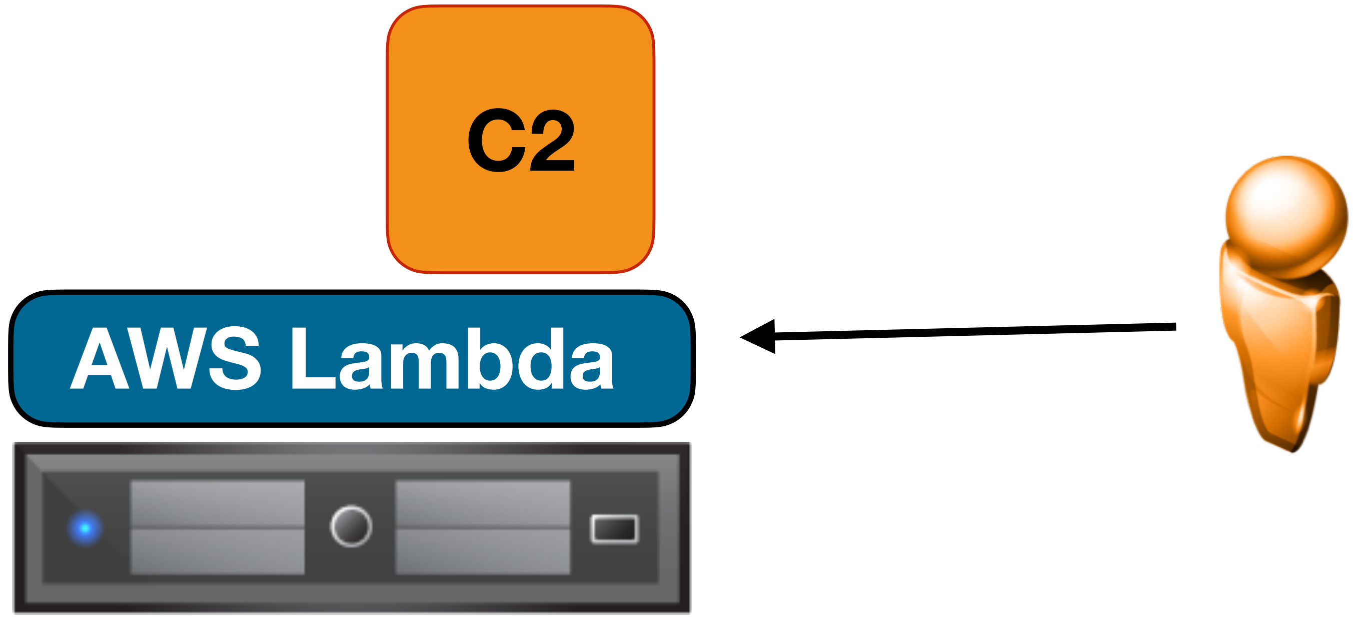


Serverless Startup

AWS Lambda

- 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

Clean up old containers once not in use



Serverless Demo

Serverless Pros/Cons

Benefits:

- Can be cheaper
- No server/vms/containers/OS management
- Very easy scalability
- Very easy to deploy

Drawbacks:

- Security issues? Less control. Less isolation?
- Slower - first request to serverless is very slow (Cold start)
- Vendor lockin
- Can’t run long running tasks (> 5 min)
- App must be Stateless (interact with a remote database)
- Development can be harder to debug
- Easy deployment can lead to sloppy deployment

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

This Semester

I hope you have…

- become more comfortable with at least one new language
- gained a deeper understanding of how networks work
- gotten some hands-on experience with cloud services
- gown an appreciation for the challenges of distributed systems

What did you learn?

Networking

- UDP vs TCP
- Socket programming
  - In the future: http libraries, Remote Procedure Calls
- Network Layers
- Performance, Throughput vs Latency, scalability
- Concurrency models: threading vs event based, etc
  - Lightweight threading models (go routines)

What did you learn?

Distributed Systems

- Reliability/Consensus - two generals, byzantine fault tolerance
  - Raft - Etcd, zookeeper, consul,
- Microservices / serverless (Architectural trends)
  - Trend: application development should be the focus instead of application deployment
- HTTP/REST - web technologies can be used for many applications
- Scale up vs Scale out

What did you learn?

Cloud Computing

- Load Balancing - ties back to networking layers
- AWS is not a total monopoly
  - Try some other services!
- Containers vs VMs
  - Surprise! I sort of lied to you. AWS Lambda is built on super fast, super lightweight virtual machines (Firecracker)
- Hybrid Cloud - combine your own servers with cloud servers