SRV409

A Serverless Journey: Under the Hood of AWS Lambda

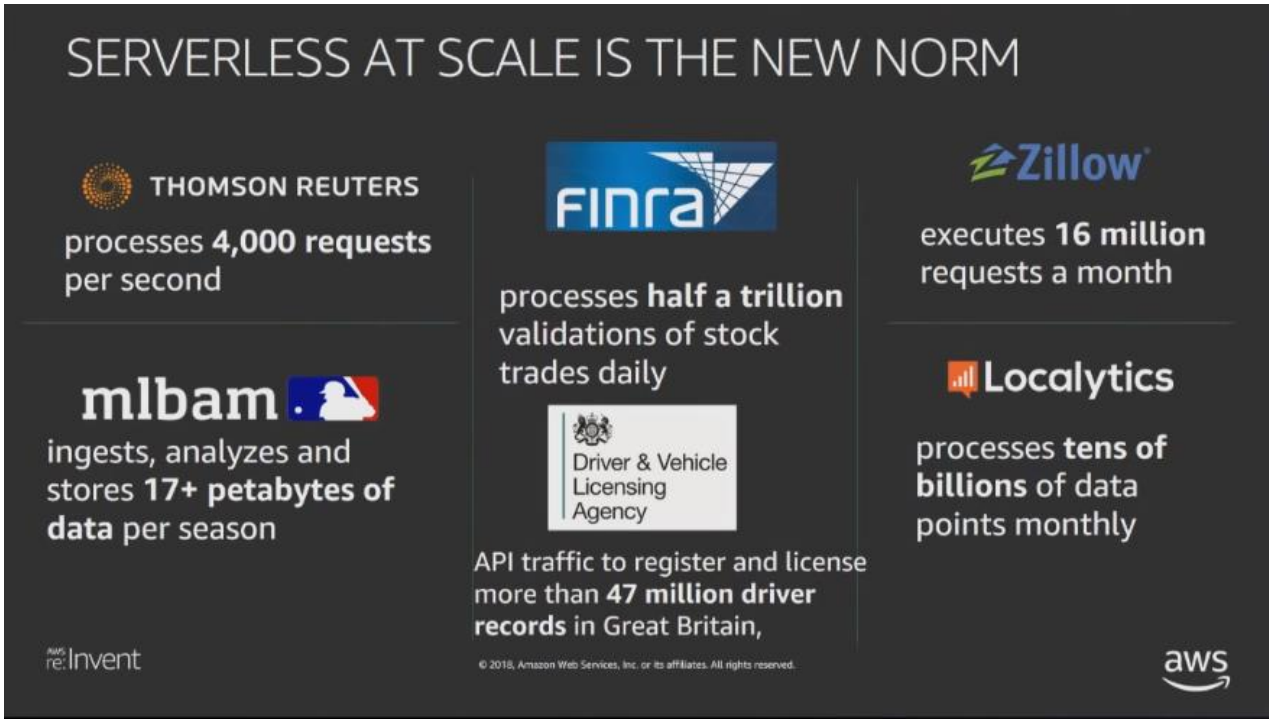
Marc Brooker
Senior Principal Engineer
Serverless

Holly Mesrobian
Director of Engineering
AWS Lambda

re:Invent

aws

Serverless computing allows you to build and run applications and services without thinking about servers. Serverless applications don't require you to provision, scale, and manage any servers. However, under the hood, there is a sophisticated architecture that takes care of all the undifferentiated heavy lifting for the developer. Join Holly Mesrobian, Director of Engineering, and Marc Brooker, Senior Principal of Engineering, to learn how AWS architected one of the fastest-growing AWS services. In this session, we show you how Lambda takes care of everything required to run and scale your code with high availability.



SERVERLESS AT SCALE IS THE NEW NORM

THOMSON REUTERS processes **4,000 requests** per second

mlbam ingests, analyzes and stores **17+ petabytes of data** per season

FINRA processes **half a trillion** validations of stock trades daily

Driver & Vehicle Licensing Agency: API traffic to register and license more than **47 million driver records** in Great Britain,

Zillow executes **16 million** requests a month

Localytics processes **tens of billions** of data points monthly

re:Invent

aws

Running
Highly Available
Large Scale
Systems Is a Lot of Work

Load
Balancing

re:Invent

liates. All rights reserved.

aws

You need to have load balancing in every layer of your architecture for scalability and availability. You also need to configure the needed routing rules.

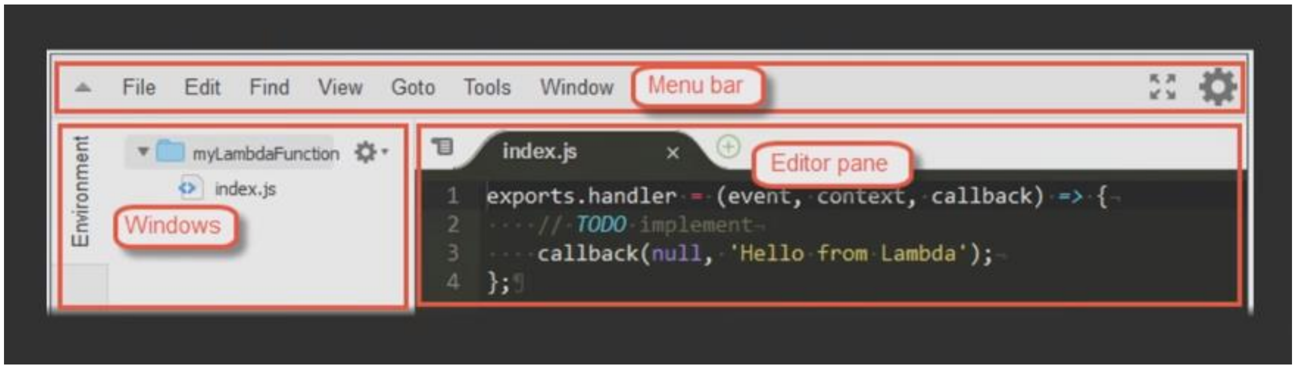
Scaling
Up and
Down

re:Invent

liates. All rights reserved.

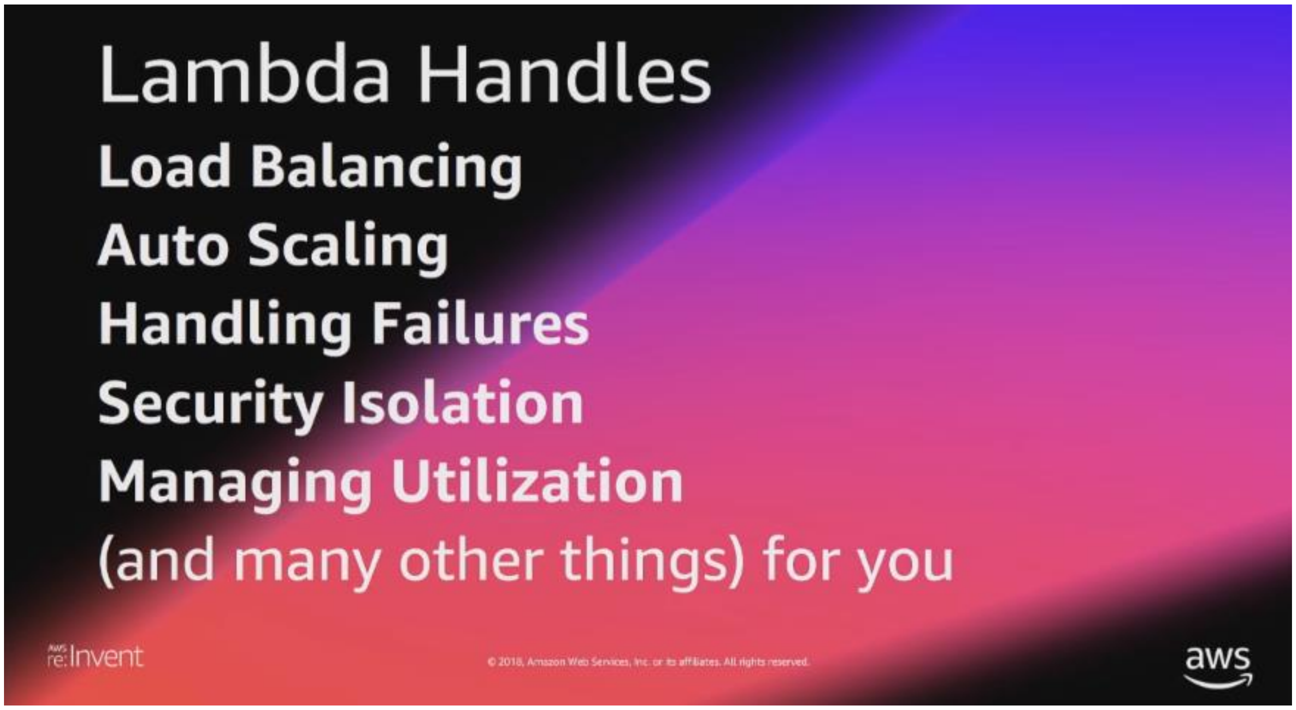
aws

You need to consider when hosts fail by configuring health-checks



```
exports.handler = (event, context, callback) => {
    // TODO implement
    callback(null, 'Hello from Lambda');
};
```

Lambda takes care of admin duties for you



Lambda Handles

Load Balancing

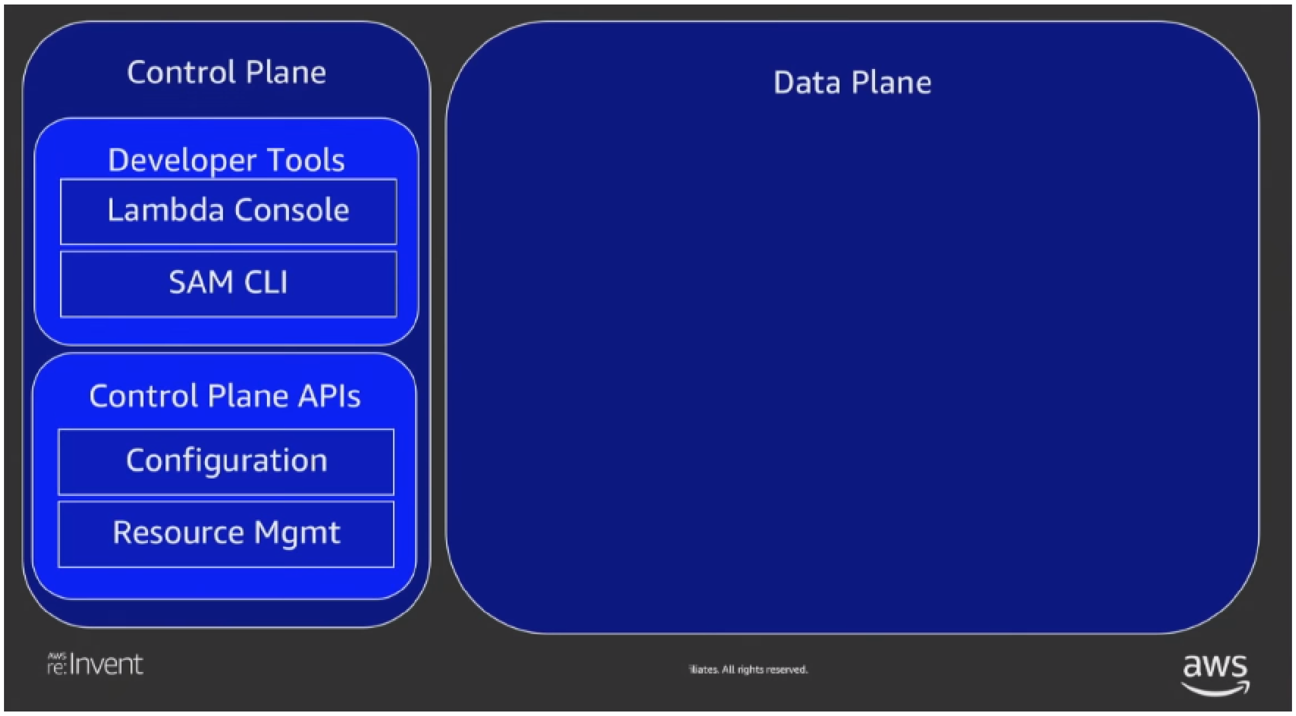
Auto Scaling

Handling Failures

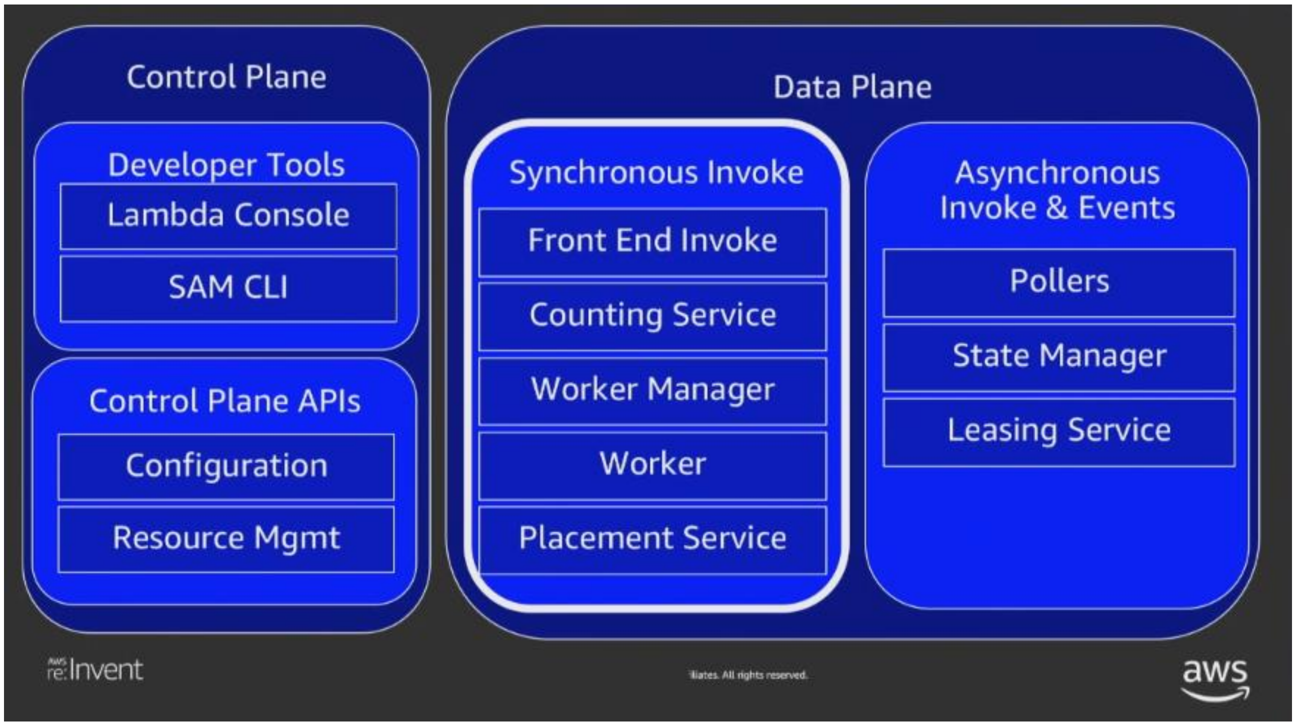
Security Isolation

Managing Utilization

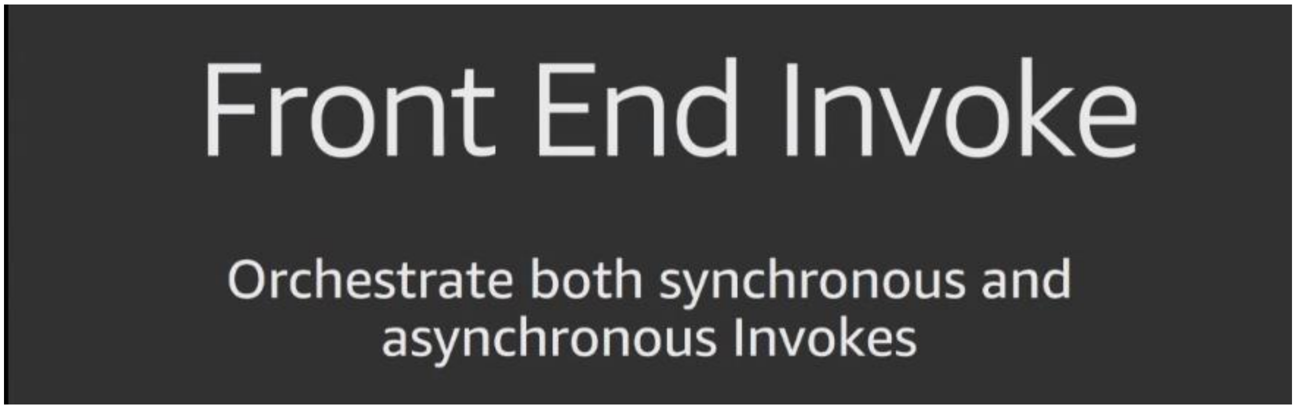
(and many other things) for you



The Lambda architecture is split into a control plane (CP) and a data plane (DP), the CP is where engineers and developers interact5 with the Lambda service using tools that end up calling the control plane APIs.

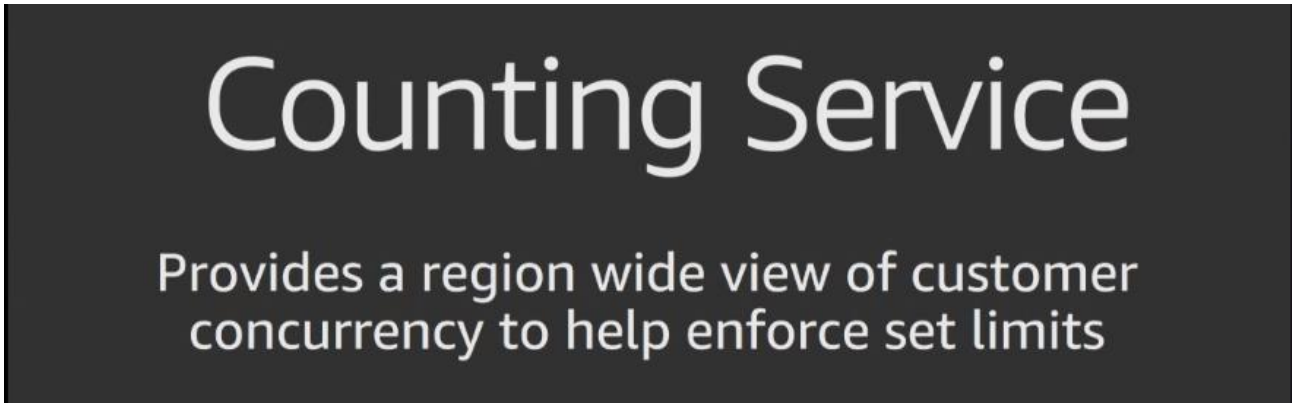


The data plane does the async invokes and inter-operates with other services that help with the event processing.


Front End Invoke

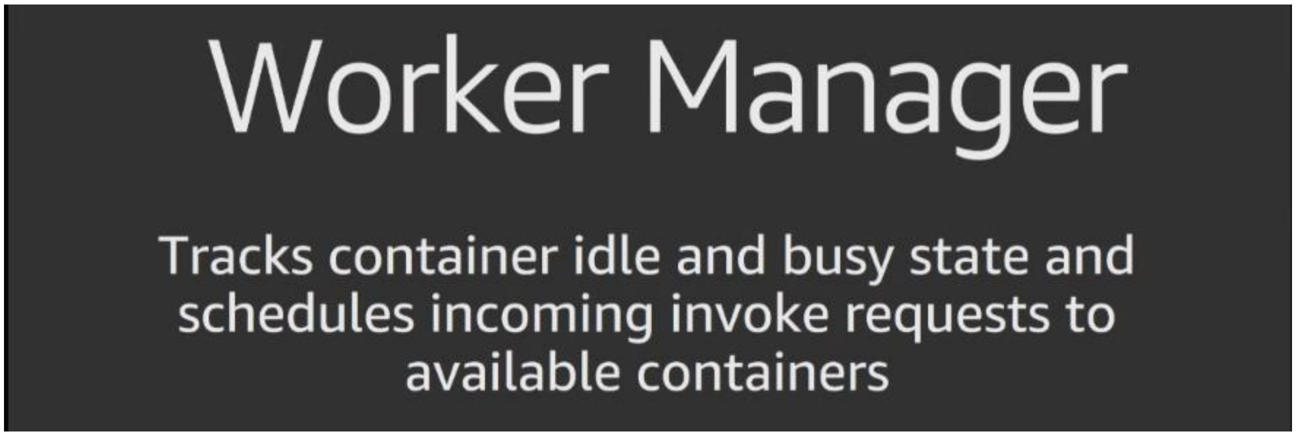
Orchestrate both synchronous and asynchronous Invokes

The first thing this service does is to authenticate the caller of the invoke function, it then retrieves and loads function metadata and code, confirms the concurrency with the counting service, then maps the function the worker manager that controls the actual worker.


Counting Service

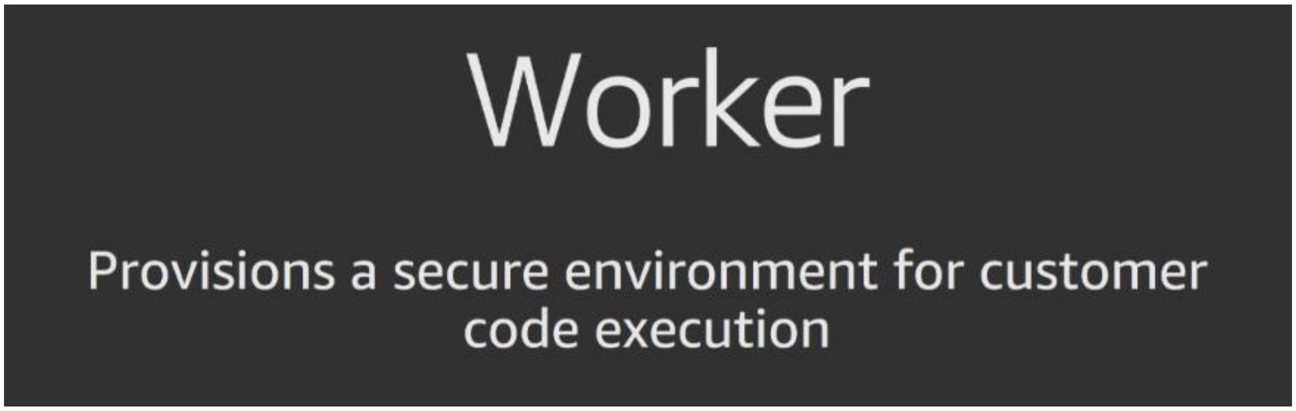
Provides a region wide view of customer concurrency to help enforce set limits

This service uses a quorum-based protocol and designed for high throughput, resilient, and HA.


Worker Manager

Tracks container idle and busy state and schedules incoming invoke requests to available containers

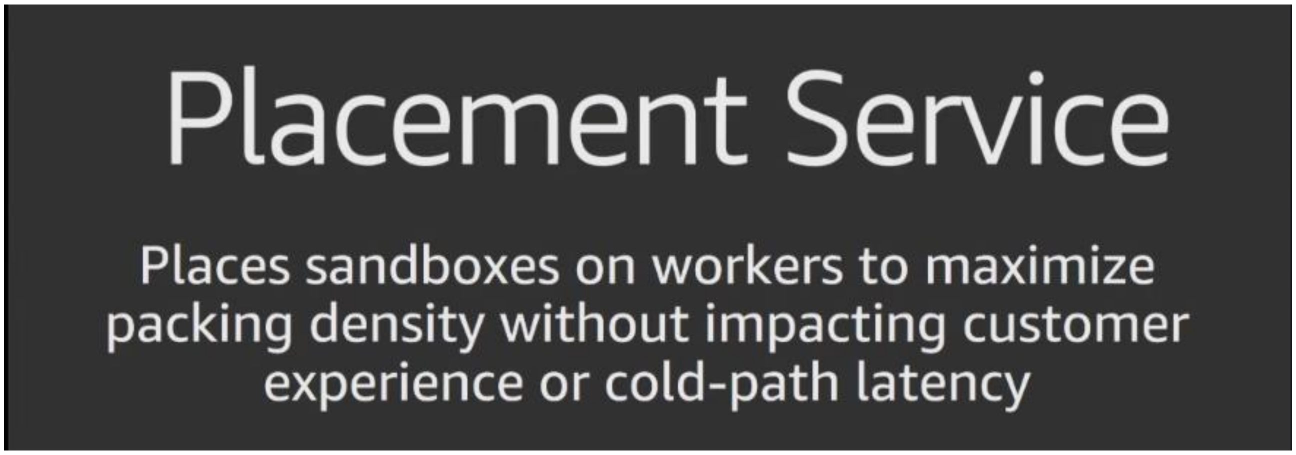
It spins up or down sandboxes to prevent waste and interacts with the placement service.



Worker

Provisions a secure environment for customer code execution

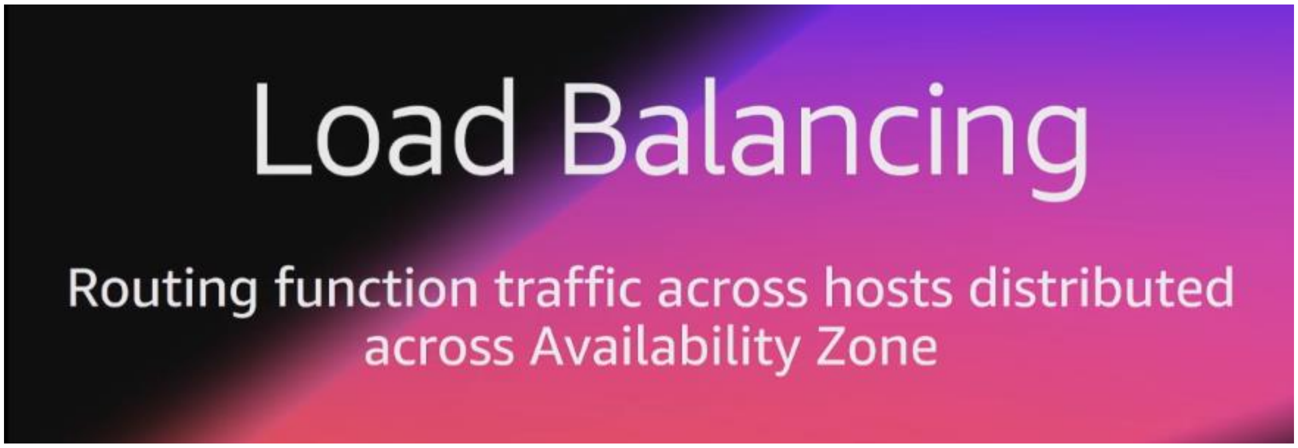
This creates and manages a collection of sandboxes needed for function executions, downloads customer code and executes them, manages communicating with other AWS services like CloudWatch.



Placement Service

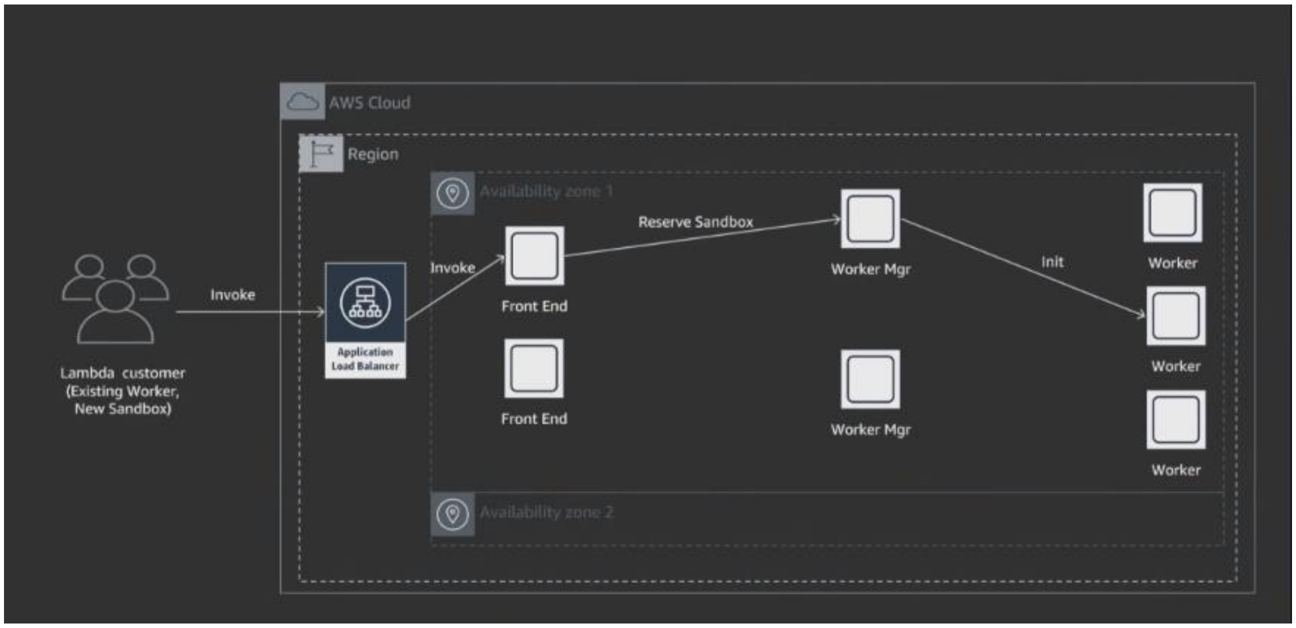
Places sandboxes on workers to maximize packing density without impacting customer experience or cold-path latency

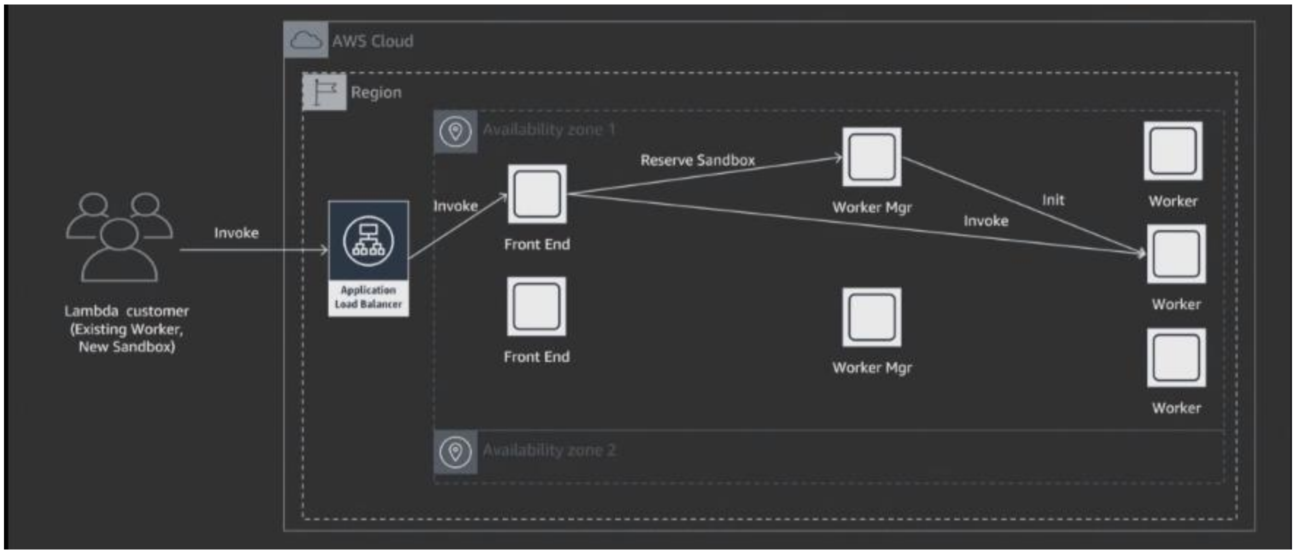
It also monitors worker health and removes bad workers.



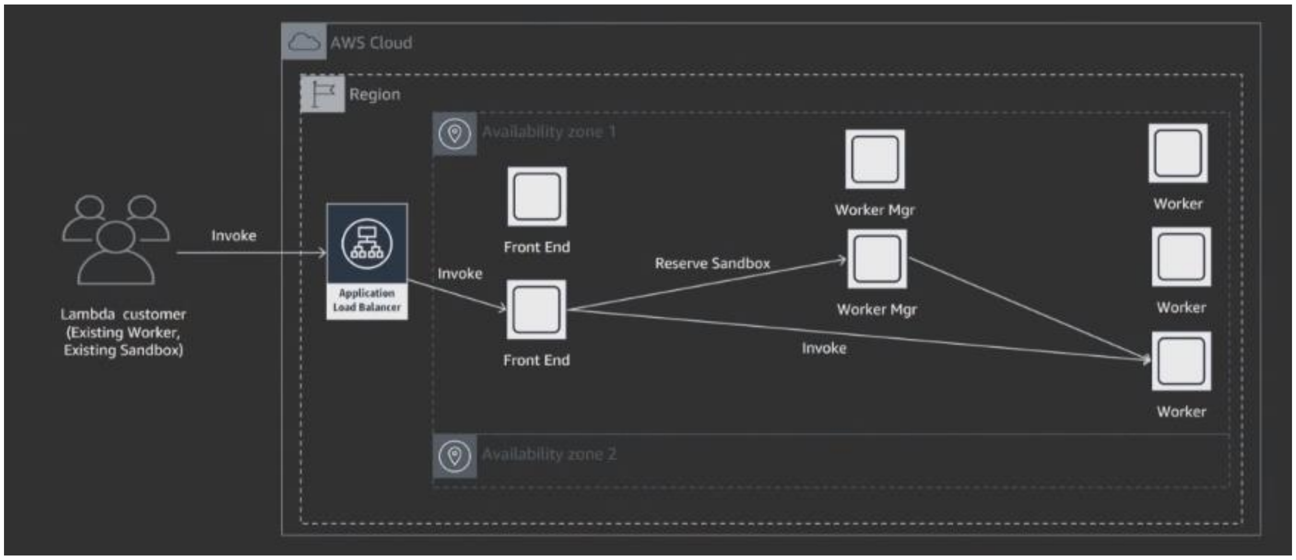
Load Balancing

Routing function traffic across hosts distributed across Availability Zone





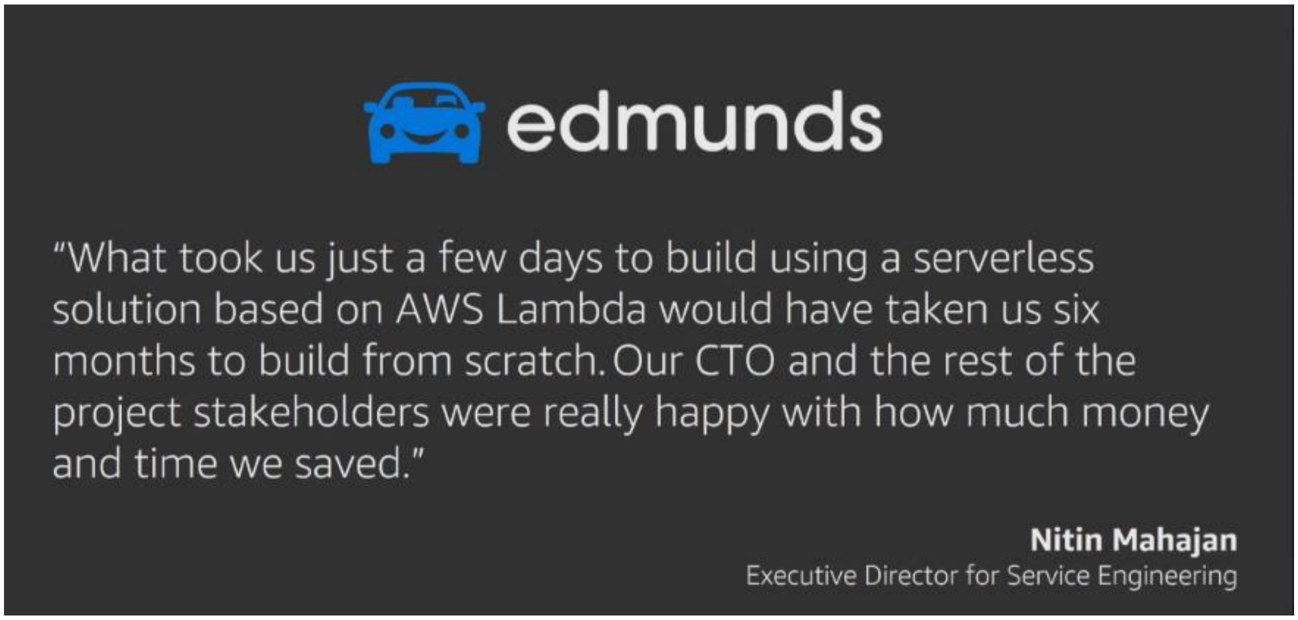
We now have a warm sandbox that we can call invoke on



Assuming we already have a warm sandbox when a call comes in, it gets faster to calling the init of your code

SERVERLESS CUSTOMERS

© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.



edmunds

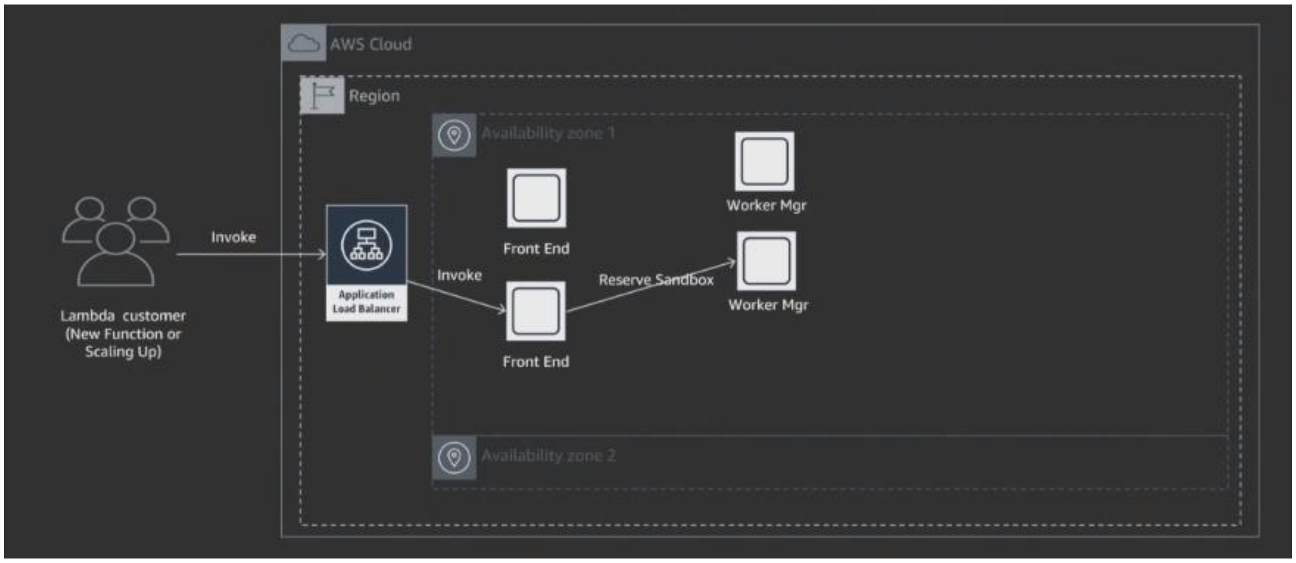
"What took us just a few days to build using a serverless solution based on AWS Lambda would have taken us six months to build from scratch. Our CTO and the rest of the project stakeholders were really happy with how much money and time we saved."

Nitin Mahajan
Executive Director for Service Engineering

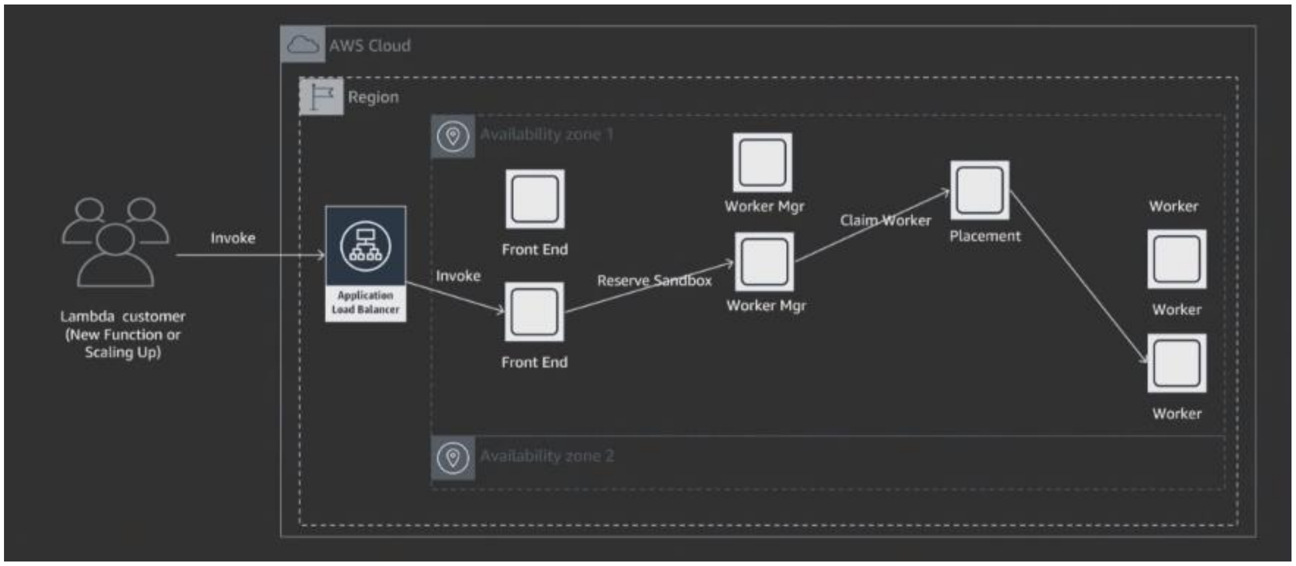
Auto Scaling

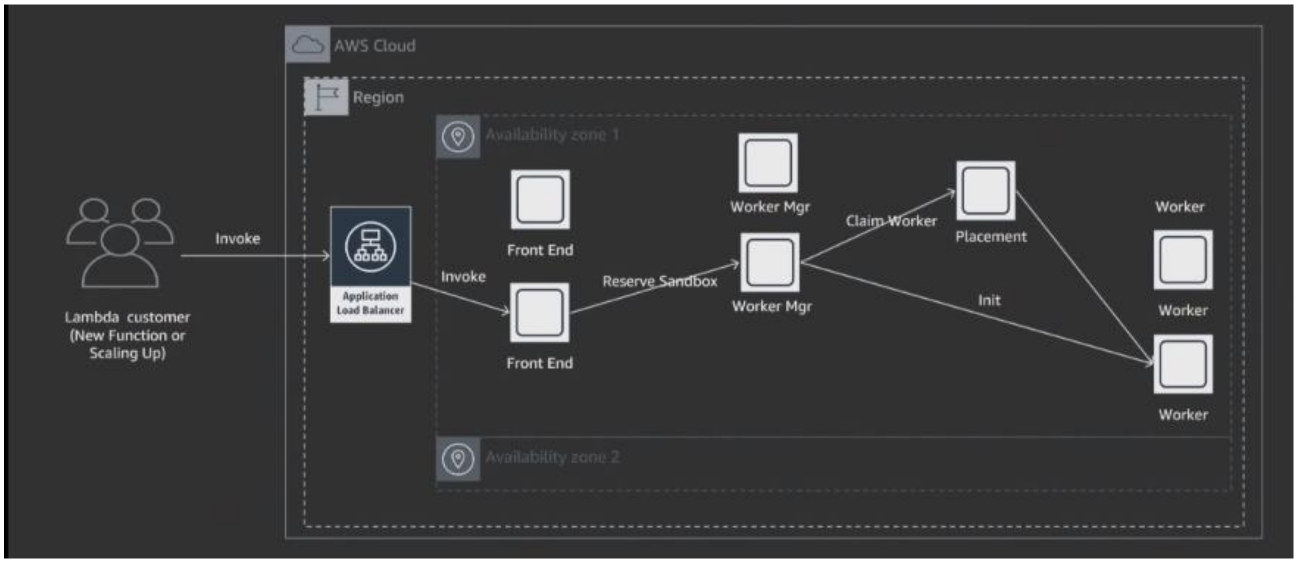
Provision function capacity when needed and releasing when not needed

What happens when we need to scale up new workers quickly?

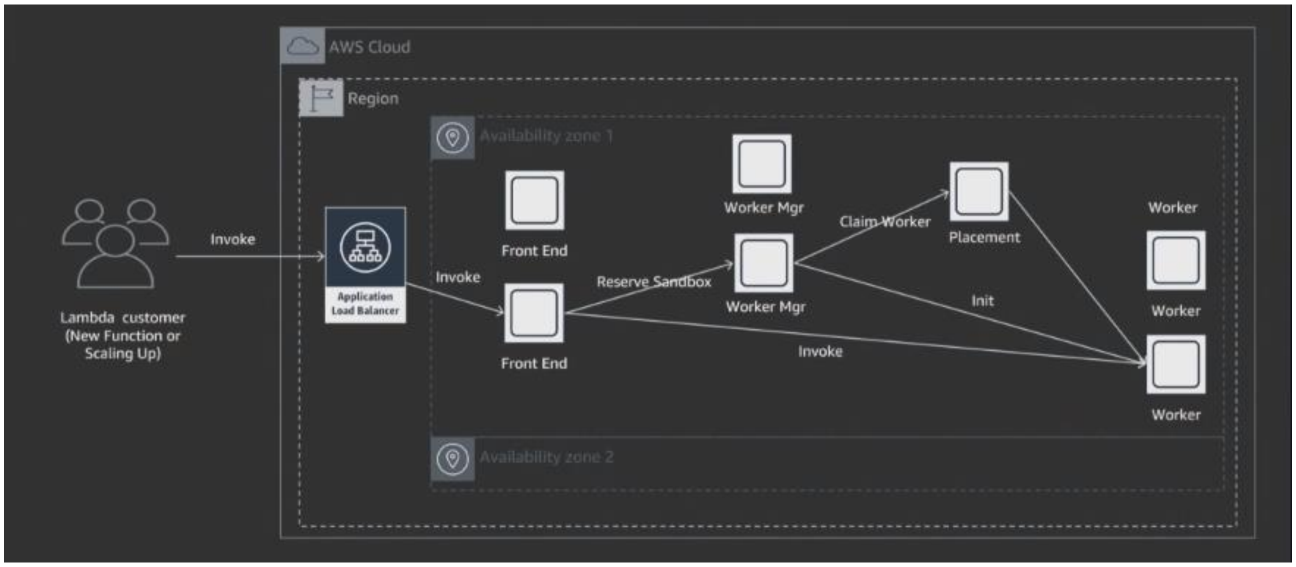


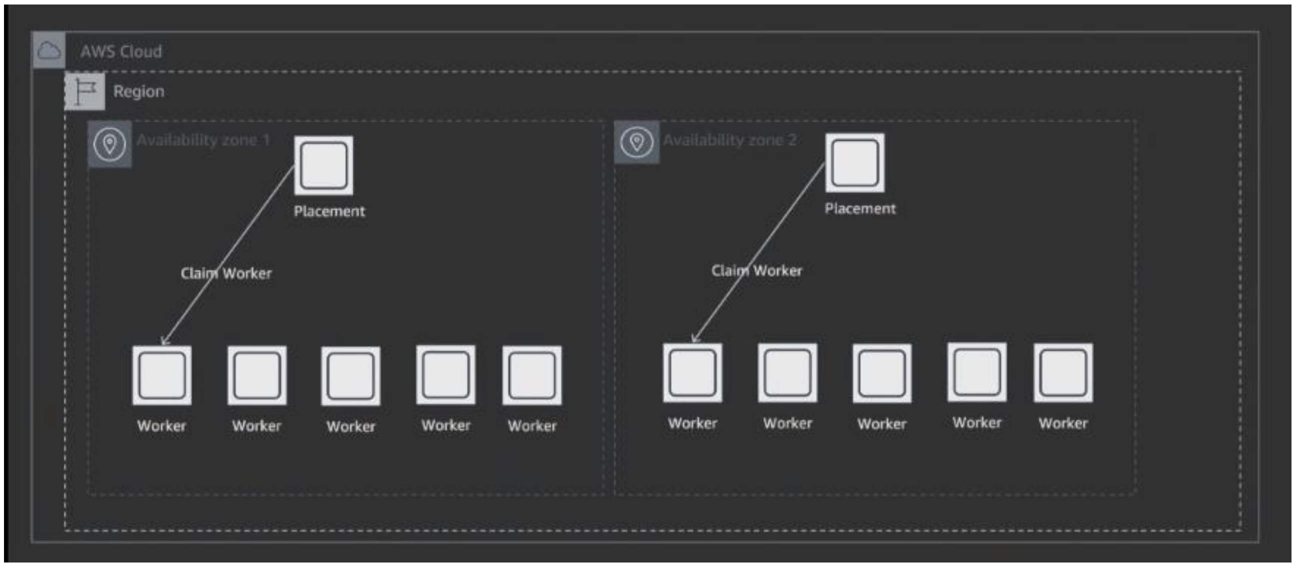
But this time the WM says it does not have a worker



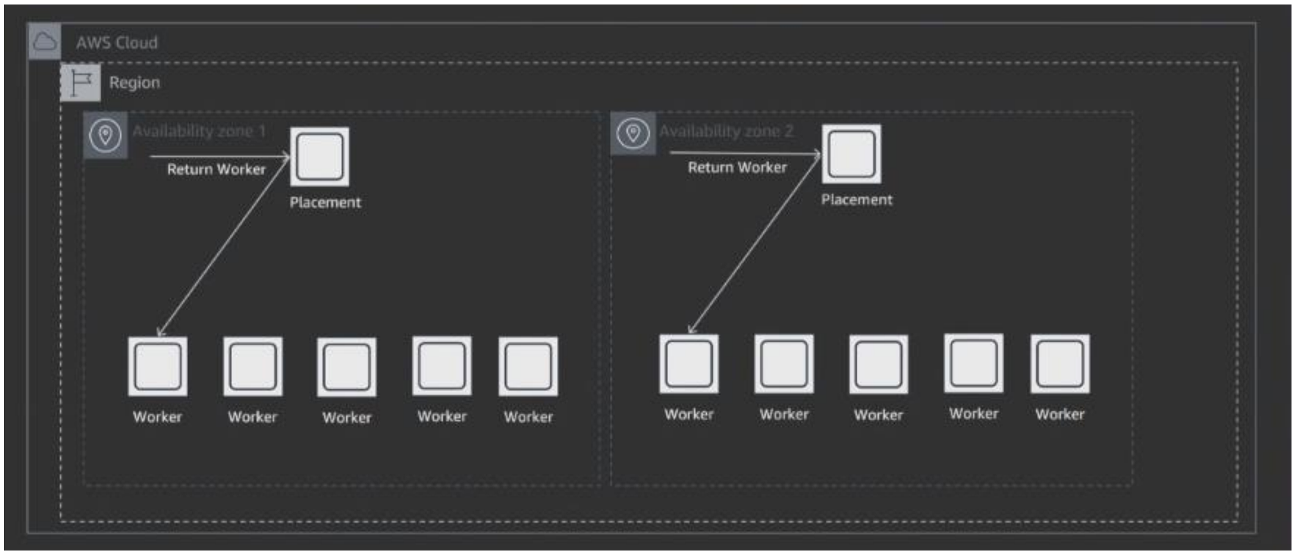


We now have a warm sandbox that the front end can call invoke on

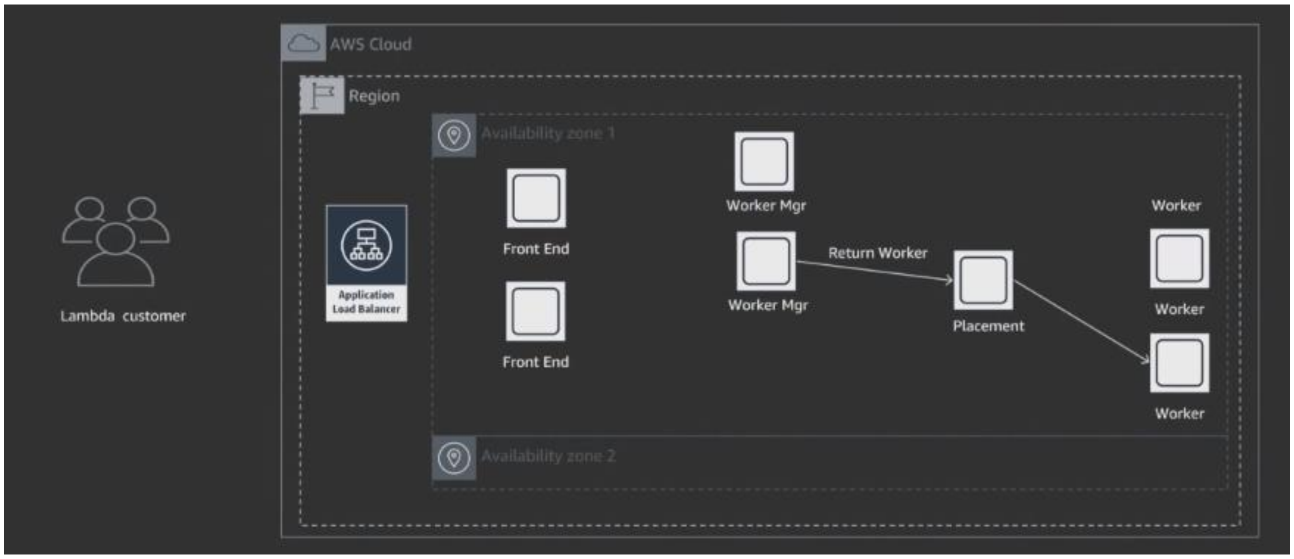




The Placement service ensures sufficient worker capacity to continue to fulfill Worker Manager requests for Worker hosts for a lease of between 6 – 10 hours.



The WM must return the Worker with an expiring lease.

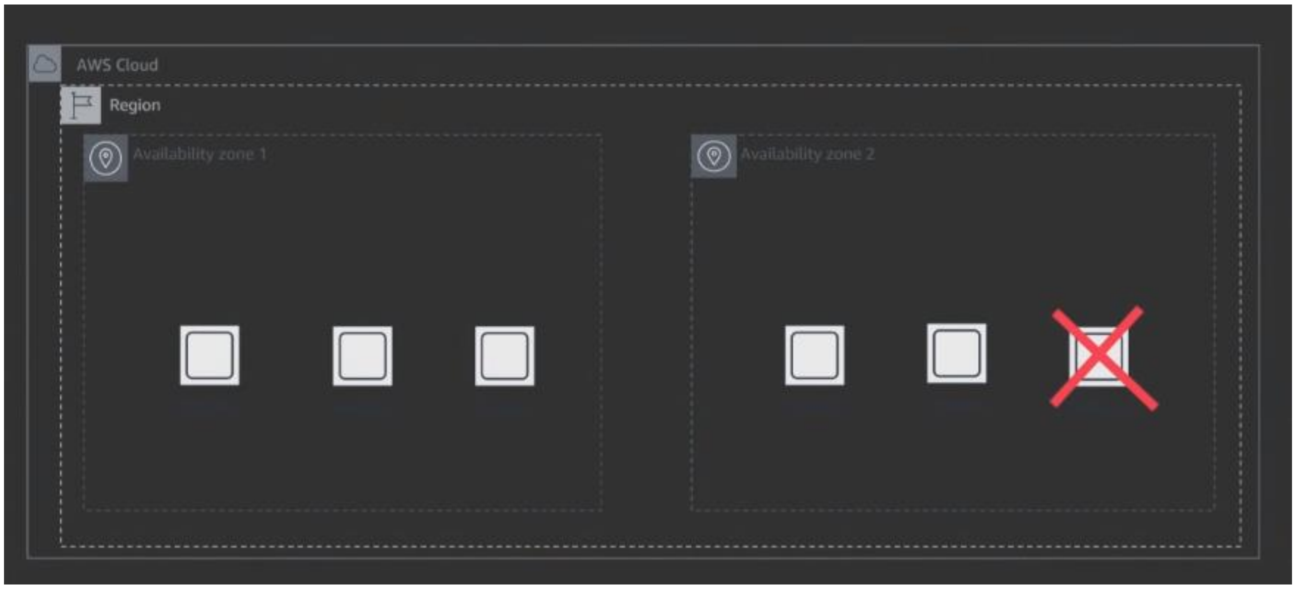


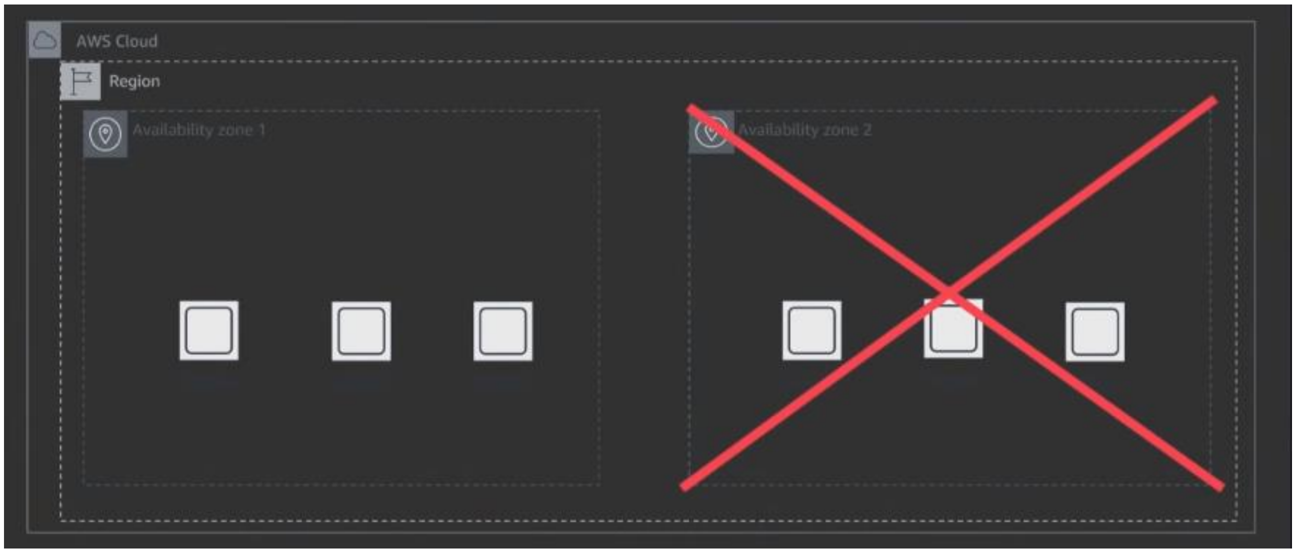
Workers are returned when their leases expire

Handling Failures

Handling Host and Availability Zone failure

With Lambda: Always have a healthy host

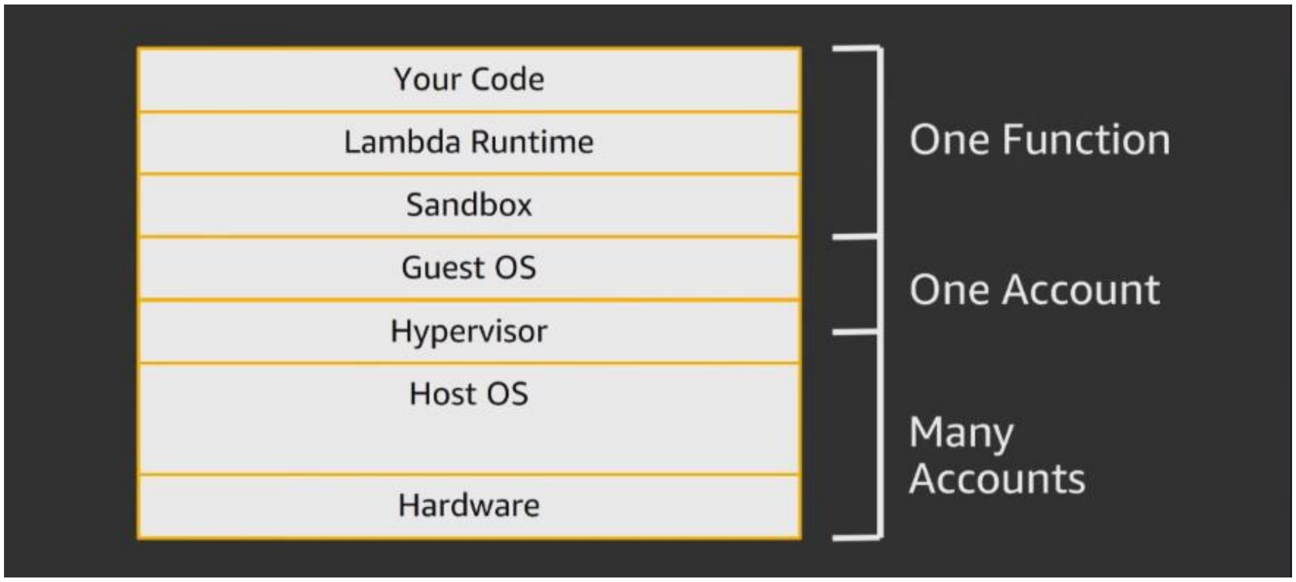




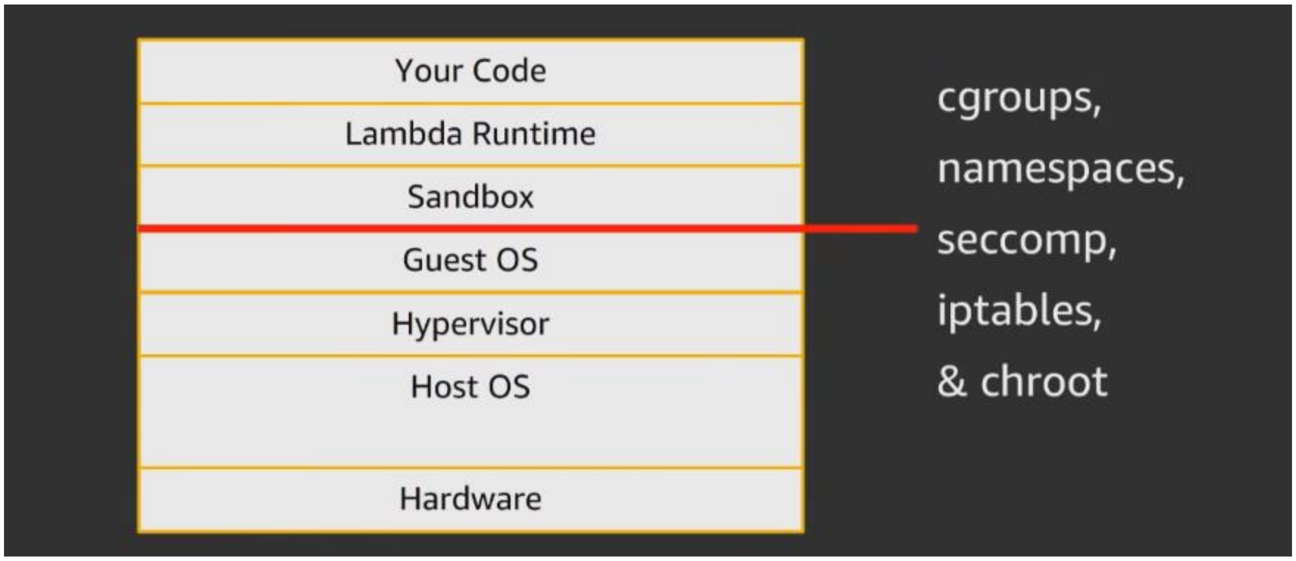
Isolation

Keeping Workloads Safe and Separate

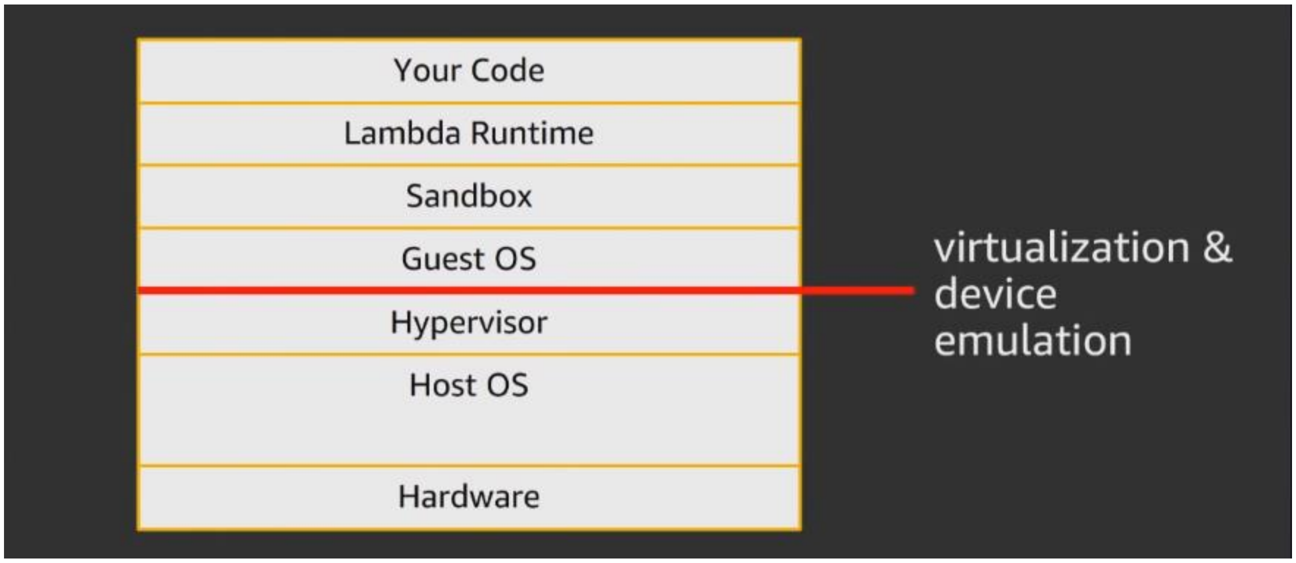
We run multiple Lambda functions on a single host at times, we keep them isolated and running with a consistent performance.

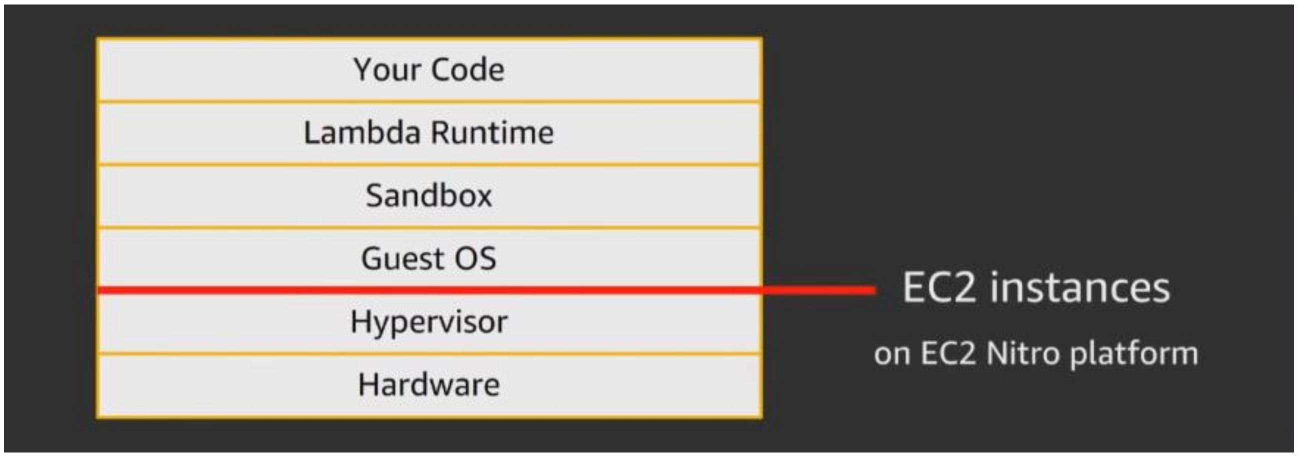


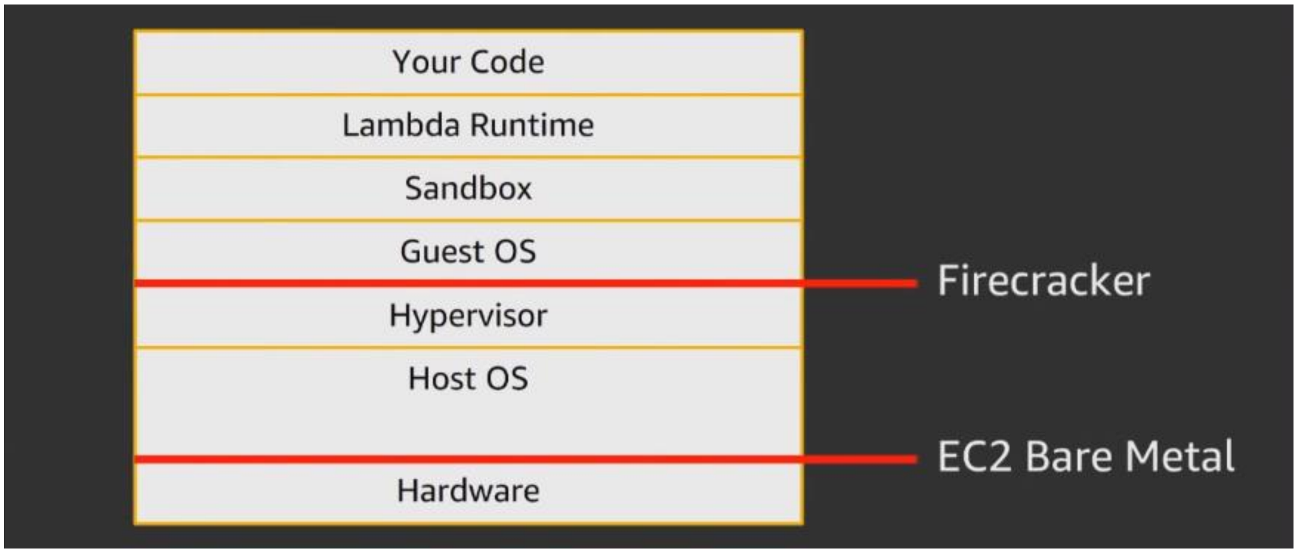
This is the stack running on a Worker node or host. The guest OS is Amazon Linux that is isolated from each other using a Hypervisor. Multiple same functions will run on the same sandbox but not overlap.



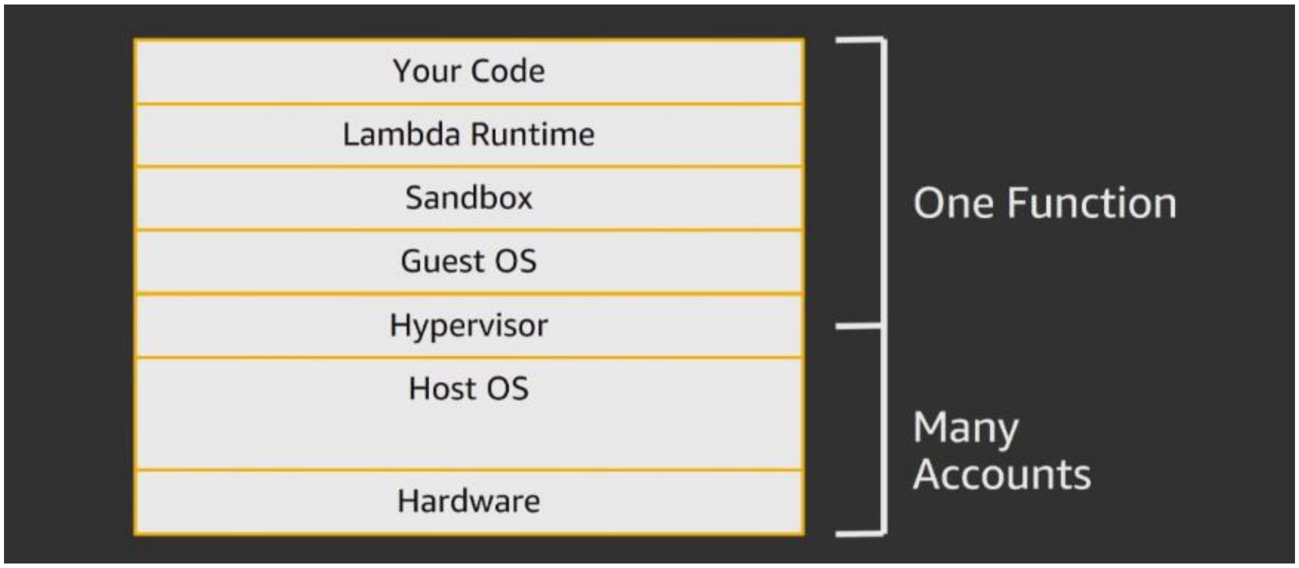
Containers are groupings of functionalities within the Linux toolbox, these include things like **cgroups**, namespaces, **seccomp**, iptables, chroot. The **cgroups** are sticky and a process added cannot get out of the cgroup. The process that your lambda function runs on is PID in its process namespace. The **seccomp** is like a firewall for your process to determine allowed **syscalls** that can be called, it helps restrict your lambda function to certain tasks.



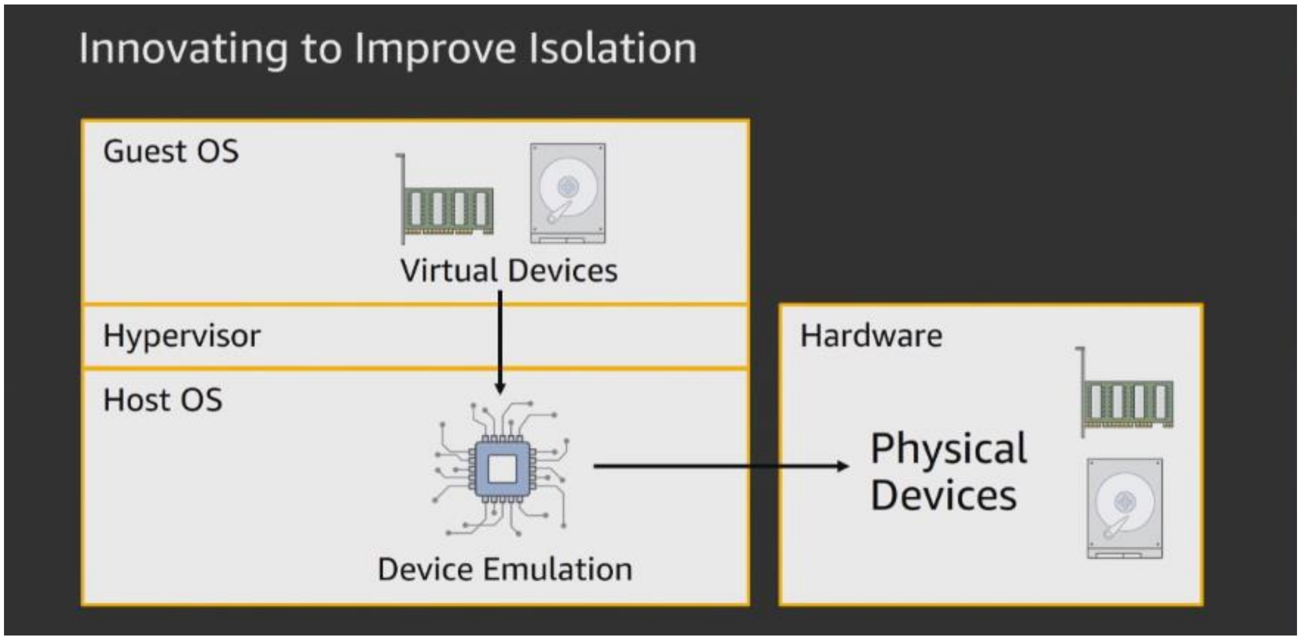




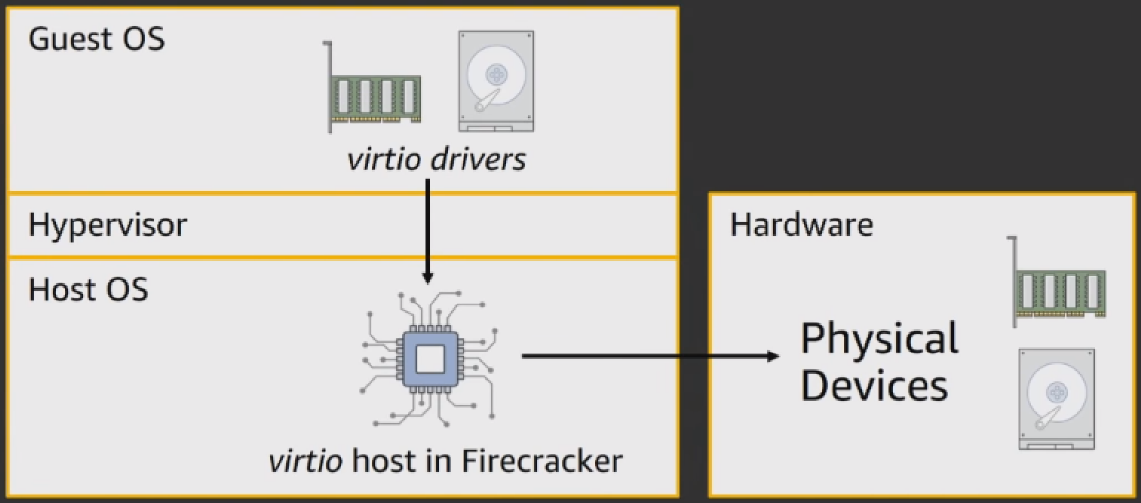
We use Firecracker to launch 100,000s of micro VMs on EC2 bare metal instances.



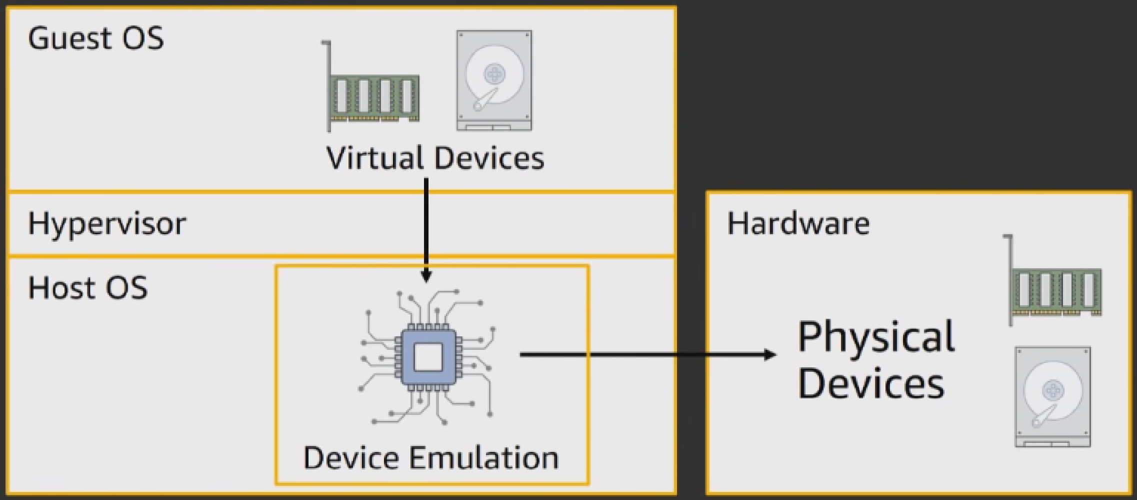
This provides strong isolation even between multiple Lambda functions when we are running in the Firecracker mode.



Innovating to Improve Isolation

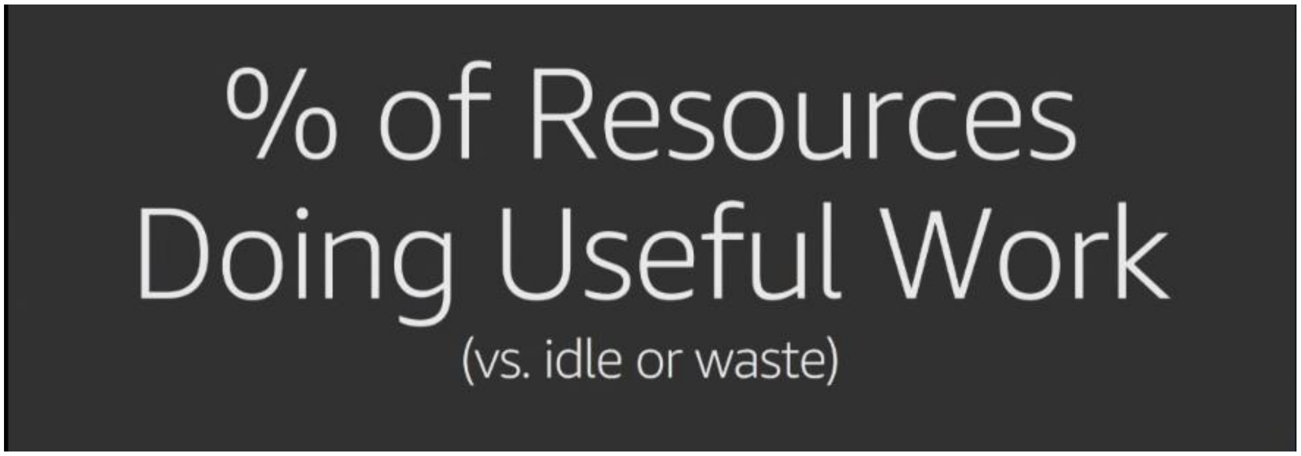


Innovating to Improve Isolation



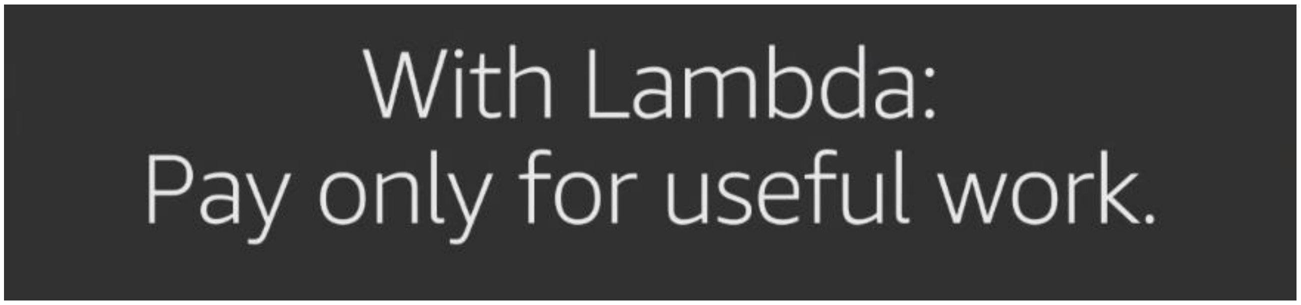
Utilization

Keeping Servers Busy

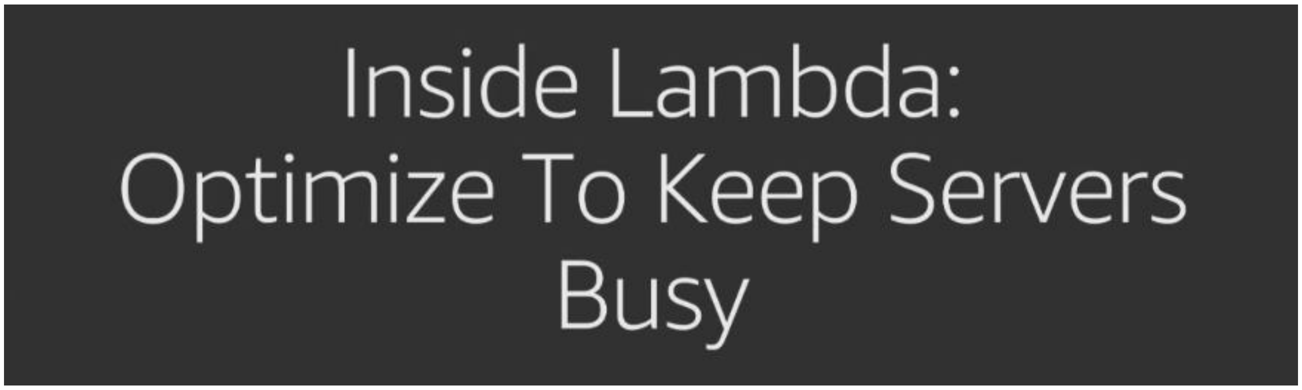


% of Resources Doing Useful Work

(vs. idle or waste)

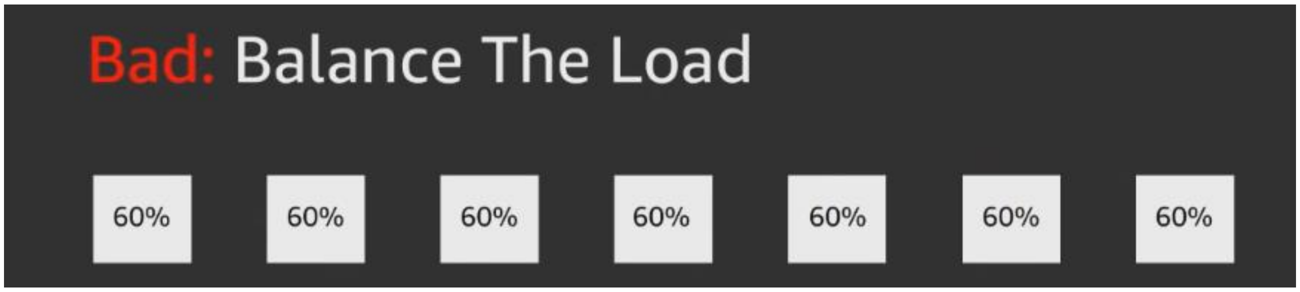


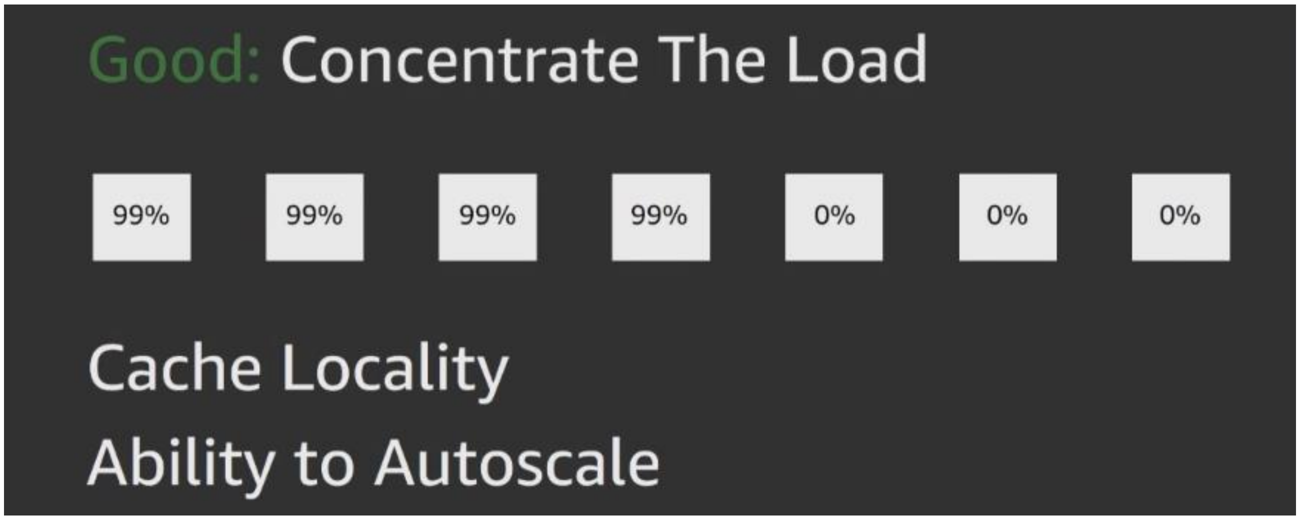
With Lambda: Pay only for useful work.

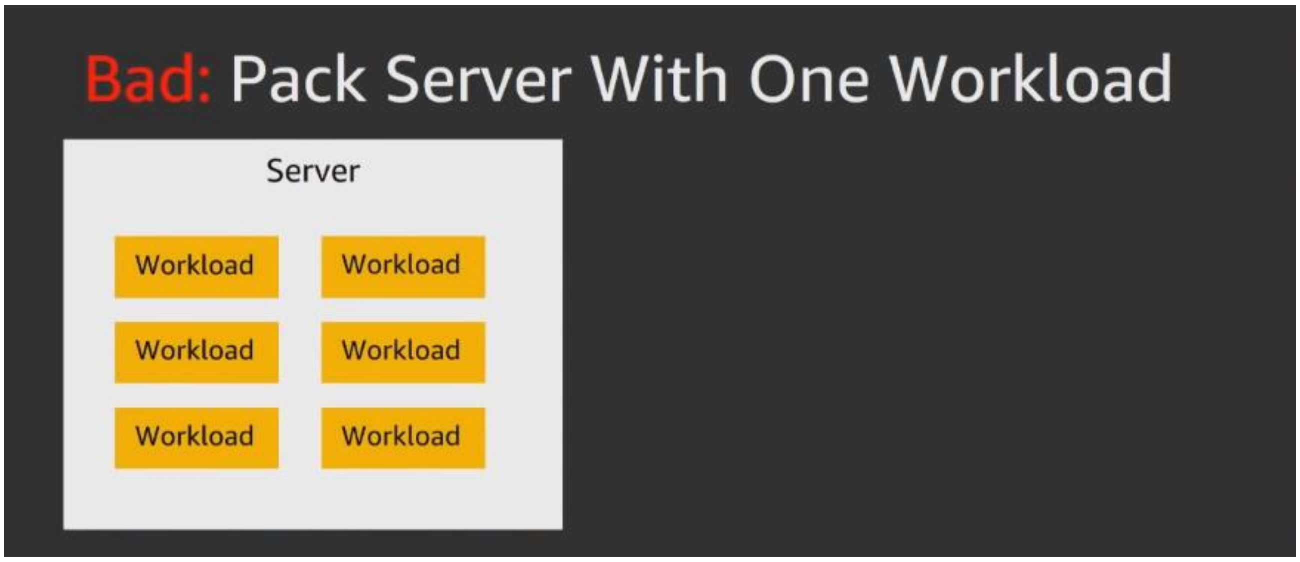


Inside Lambda: Optimize To Keep Servers Busy

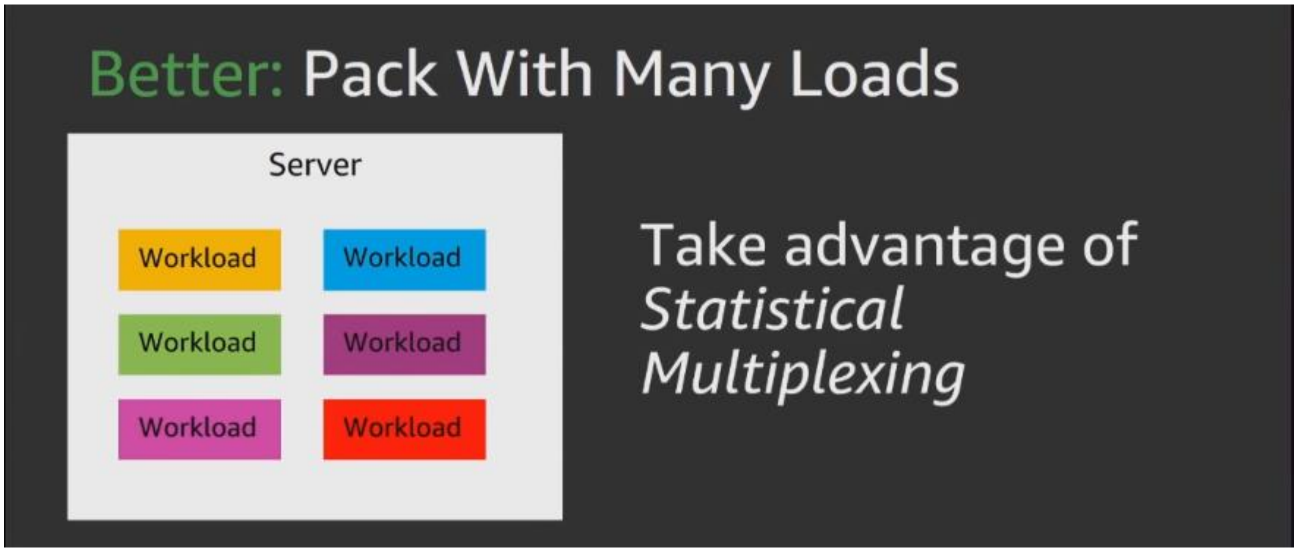
Available Sandboxes For a Function





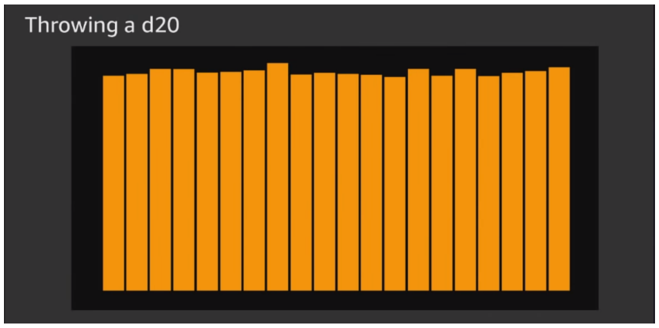


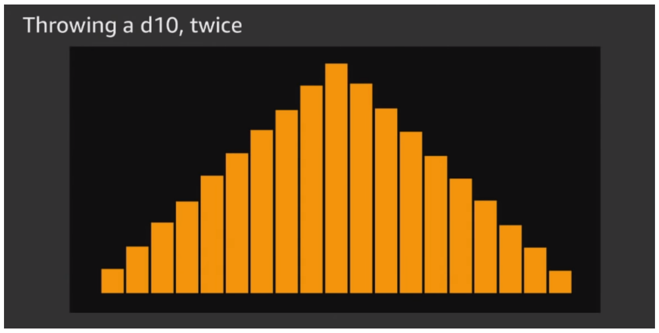
This will have very correlated load that results in same spikes or memory demands from the workloads of the same type running on the server

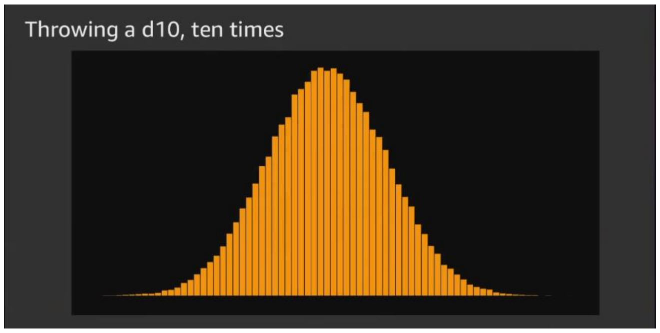


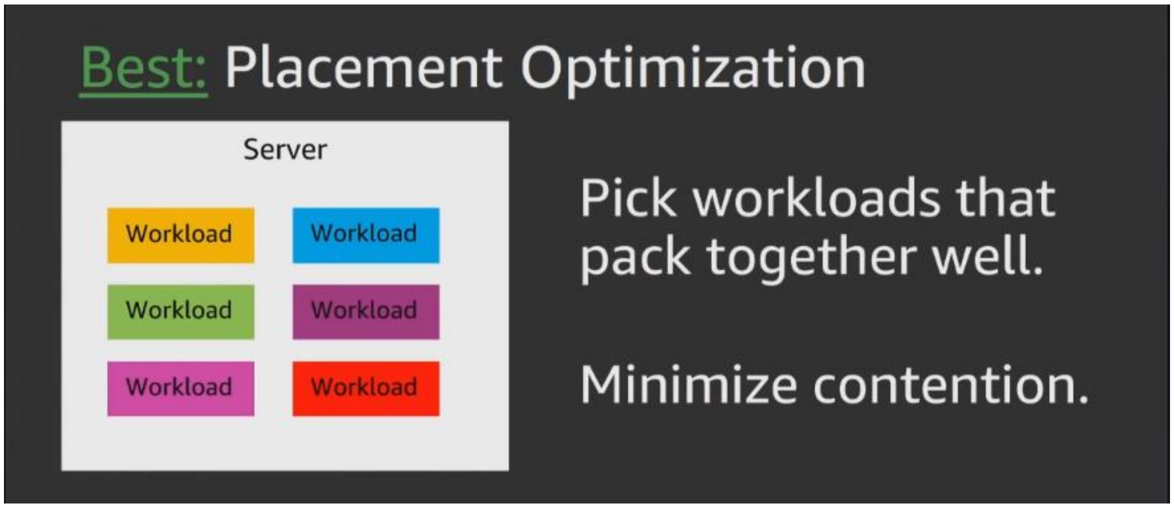
Better: Pack With Many Loads

Take advantage of *Statistical Multiplexing*





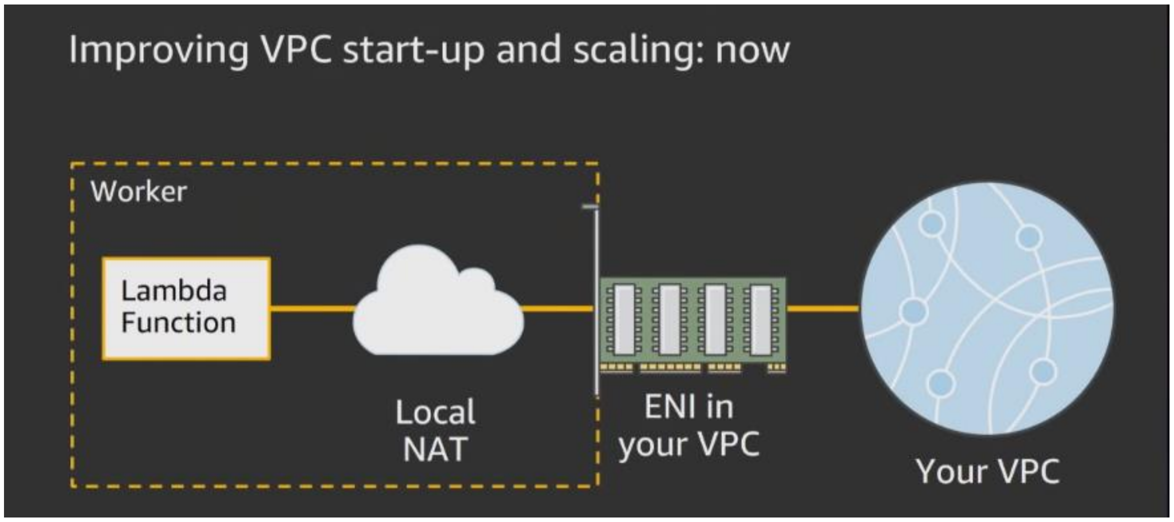




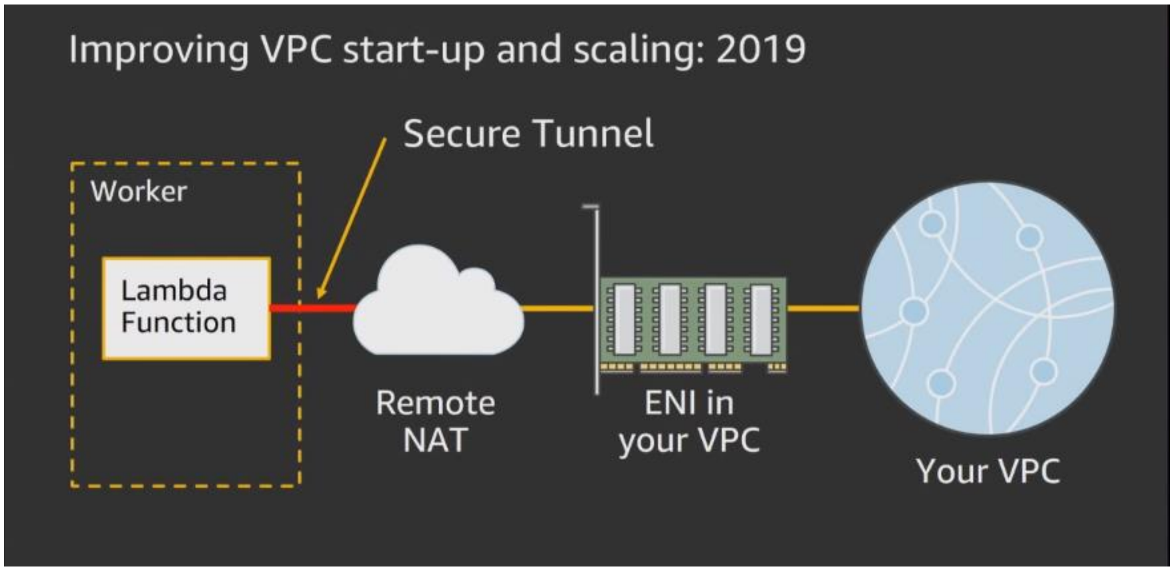
Best: Placement Optimization

Pick workloads that pack together well.

Minimize contention.

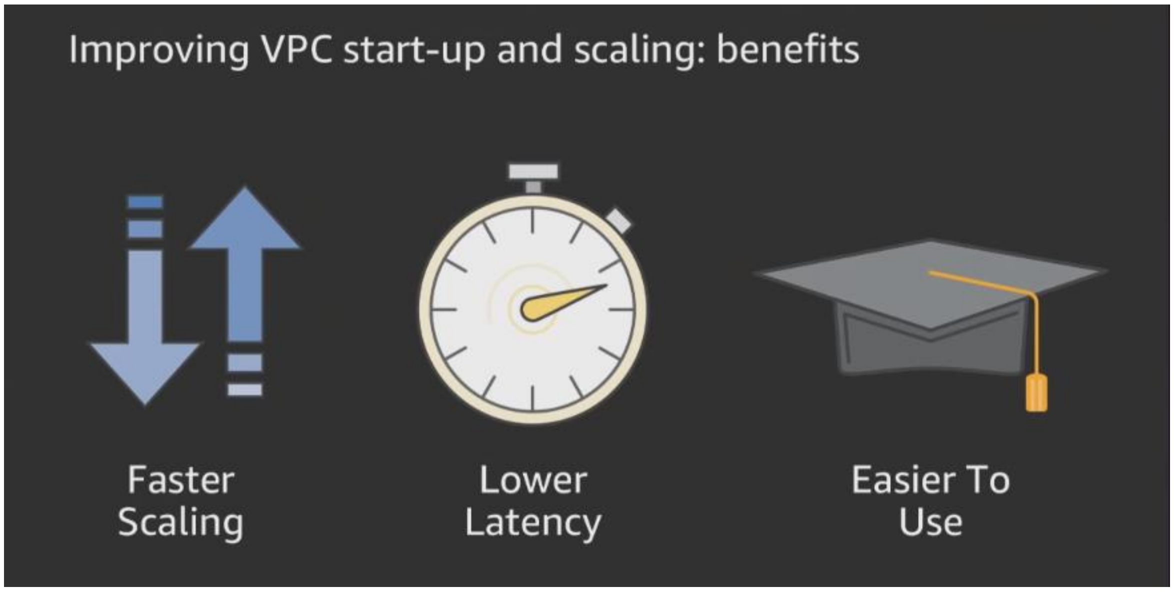


Improving VPC start-up and scaling: now



Improving VPC start-up and scaling: 2019

This allows us to use 1 ENIs to use with many lambda functions

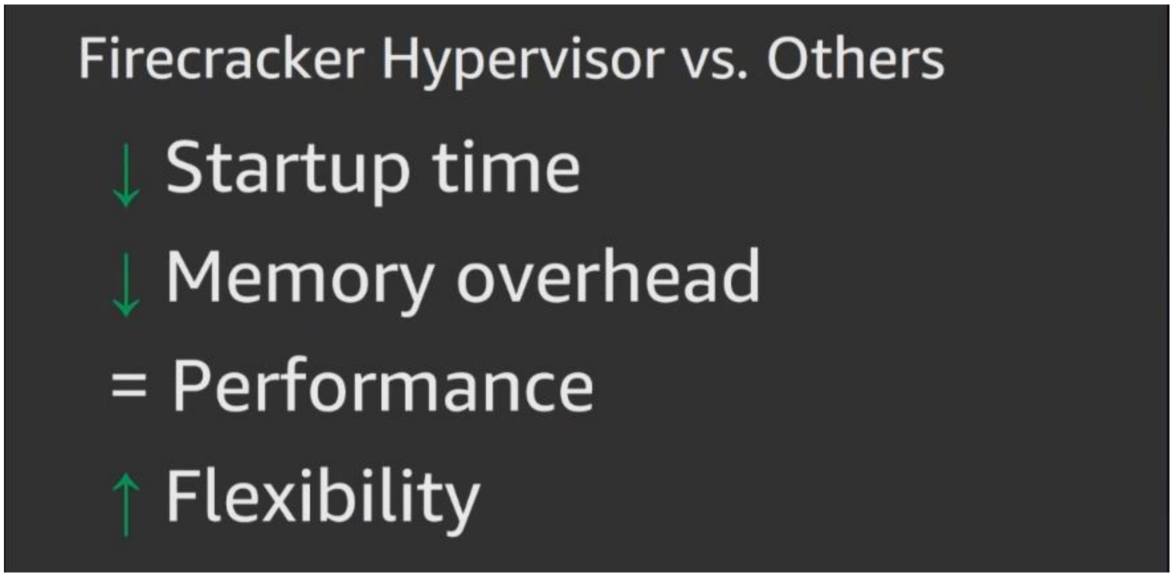


Improving VPC start-up and scaling: benefits

Faster Scaling

Lower Latency

Easier To Use



Firecracker Hypervisor vs. Others

↓ Startup time

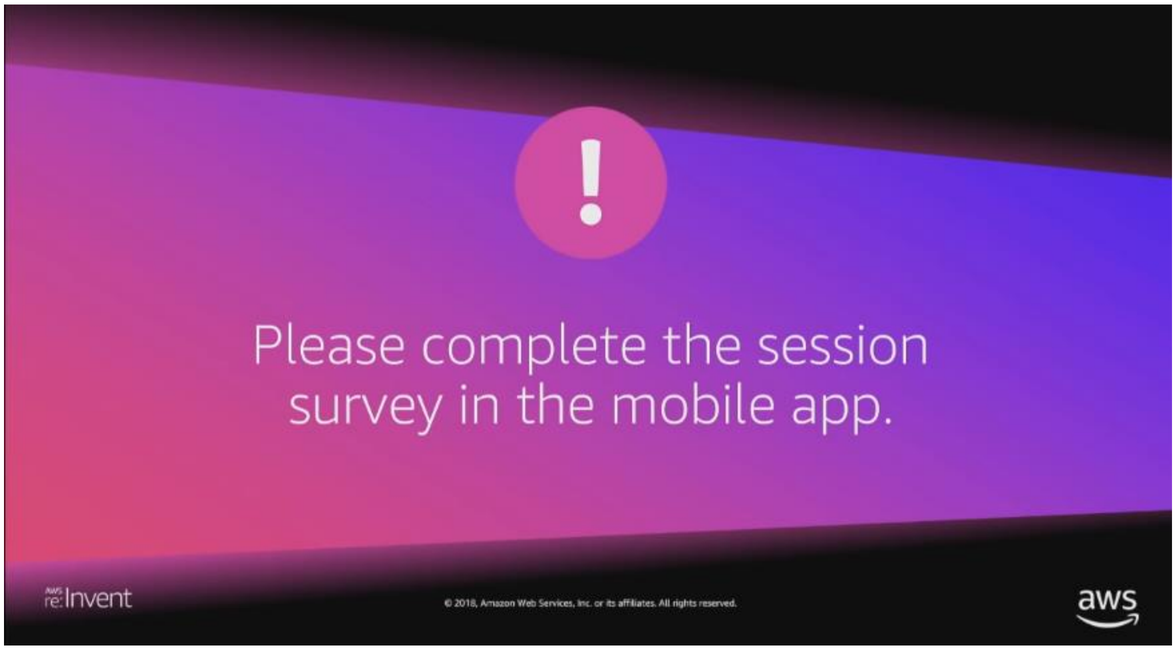
↓ Memory overhead

= Performance

↑ Flexibility

Firecracker Unlocks Higher Utilization and Scale

In Conclusion



Please complete the session survey in the mobile app.