



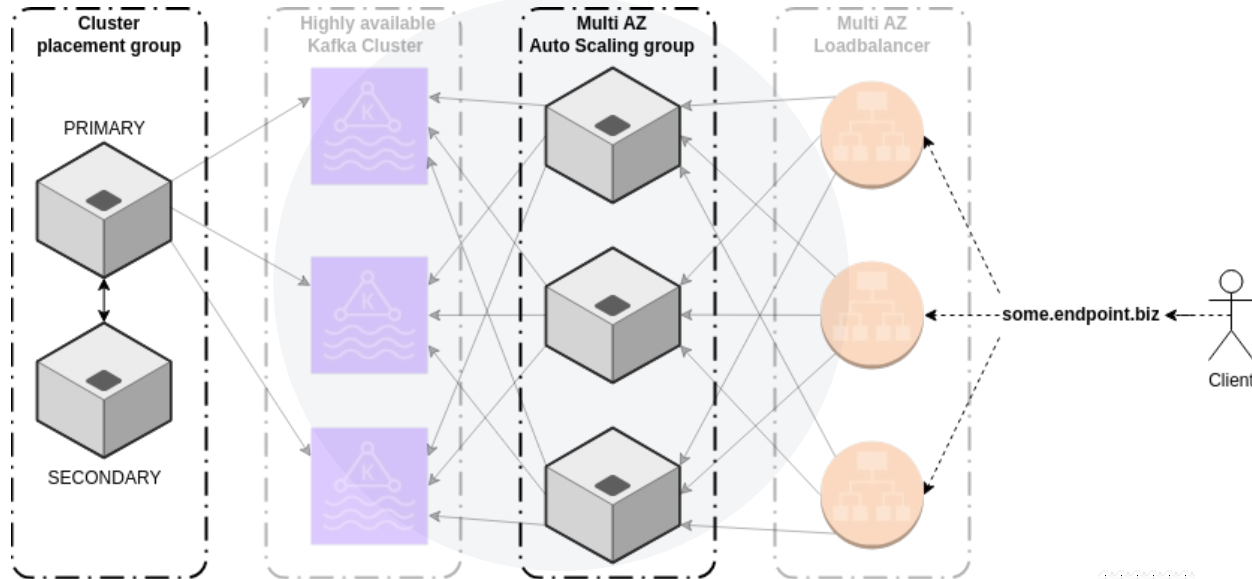
# Tips and Tricks

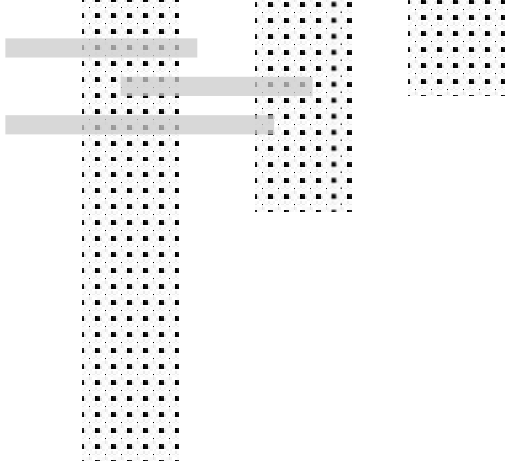
Aljaž Mežnarič

Ljubljana,  
30 May 2024

# Tips and Tricks

- **Backend**  
Placement groups
- **Frontend**  
Auto Scaling metrics
- **Client side**  
Optimal path discovery



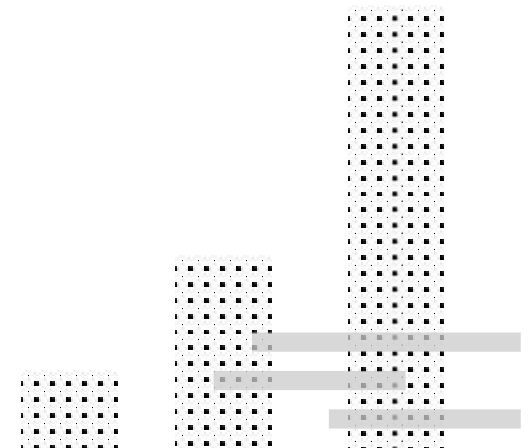
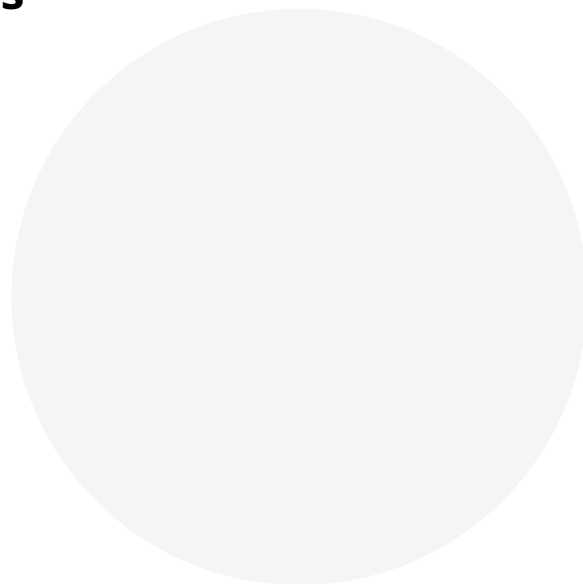


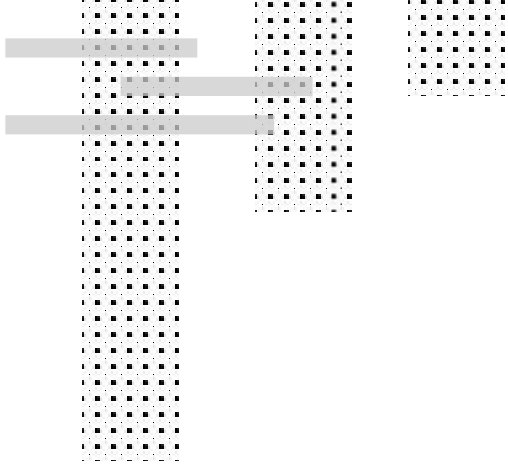
What

## AWS Placement groups



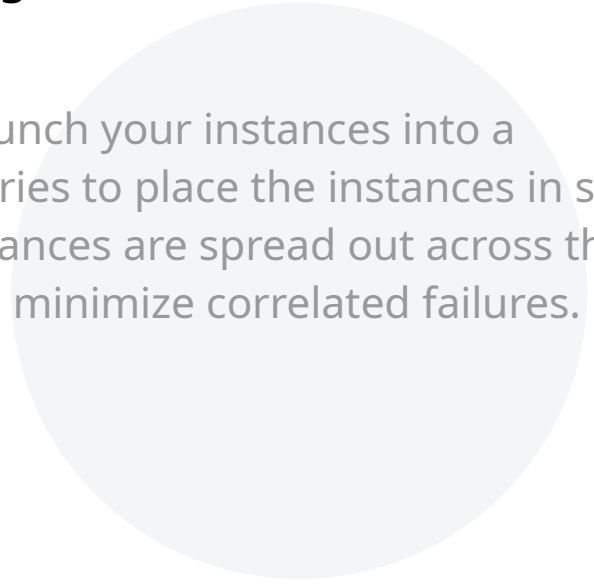
Why



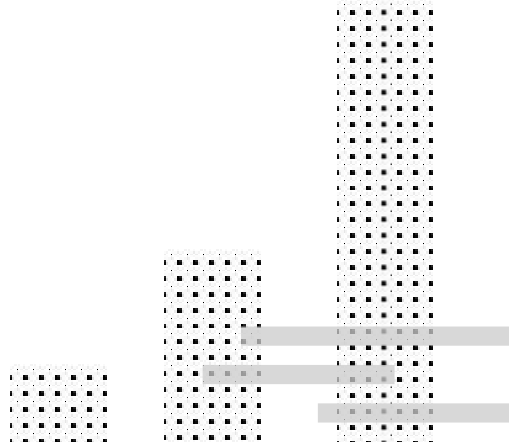


What Placement groups are a feature within Amazon EC2 that allow users to influence the placement of a group of instances to meet certain performance requirements.

## AWS Placement groups



Why When you don't launch your instances into a placement group, EC2 tries to place the instances in such a way that all of your instances are spread out across the underlying hardware to minimize correlated failures.



## Cluster placement group

focus on performance, placing instances  
close together



## Use Case

Ideal for applications that require low-latency, high-throughput network communication between instances, applications.

## Characteristics

Instances are physically located close to each other within a single Availability Zone. This proximity allows for high bandwidth and low latency.

## Limitations

Limited to a single rack in an Availability Zone

## Partition placement group

enhance fault tolerance by distributing instances across several racks



## Use Case

Useful for distributed, replicated applications which require isolation of failure domains

## Characteristics

Instances are divided into logical segments called partitions, each occupying distinct sets of racks. This means that each partition has its own set of resources and failures in one partition do not affect the others

## Limitations

Can have 7 partitions per Availability Zone

## Spread placement group

limit the number of instances per rack to  
minimize failure impact



## Use Case

Suitable for applications that require high availability and need to be resilient to hardware failures, such as critical applications that need to avoid correlated failures

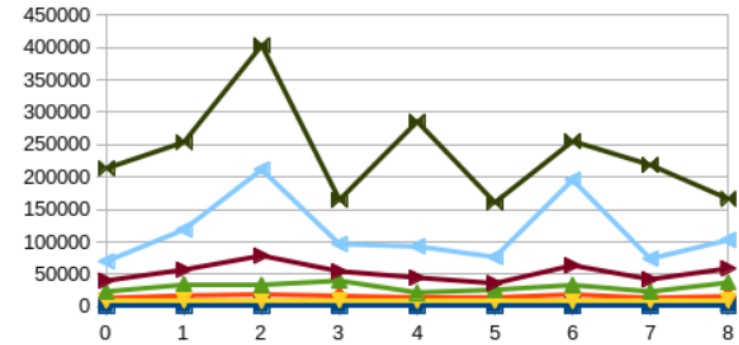
## Characteristics

Instances are distributed across different hardware within one or more Availability Zones. Each instance in a spread placement group is placed on a separate rack with its own network and power source

## Limitations

Can contain a maximum of seven running instances per Availability Zone, but can span multiple Availability Zones

## Latency profile

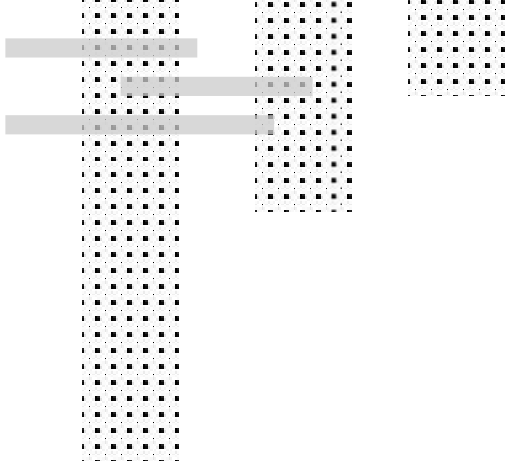


## AWS Placement groups

### Overview

Placement Group Type	Instance Types	Instance placement
cluster	Current generation (with exceptions), Previous generation (A1, C3, C4, I2, M4, R3, R4), except dedicated	Everything on 1 rack
partition	Any instance type, with dedicated only 2 partitions	Multiple instances per partition, one partition per rack, up to 7 racks
spread	Any instance type except dedicated.	One instance per rack, up to 7 racks in a single AZ. Can span multiple AZs

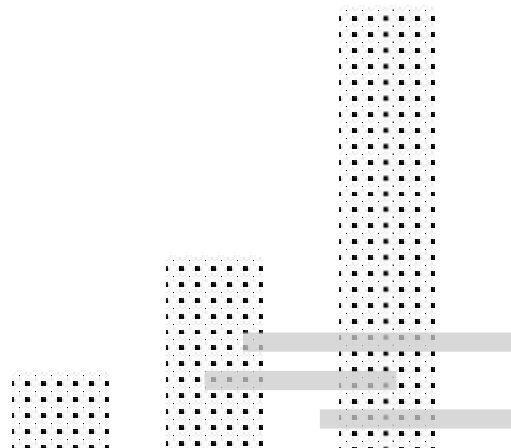
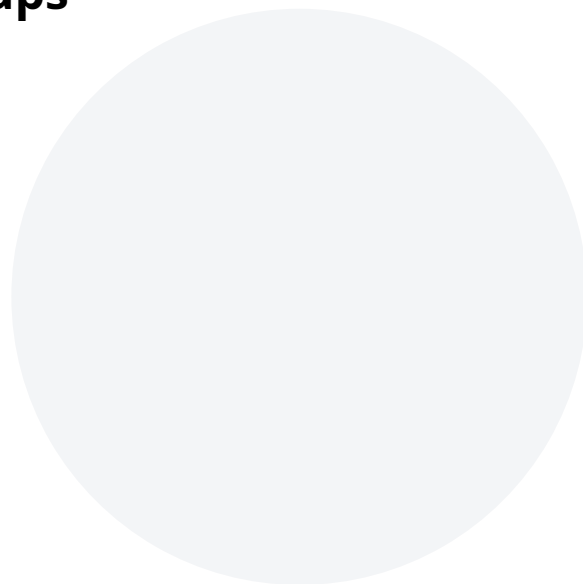


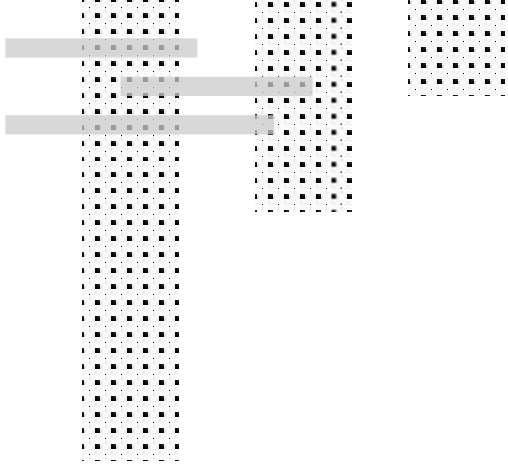


What

## AWS Auto Scaling groups


Why



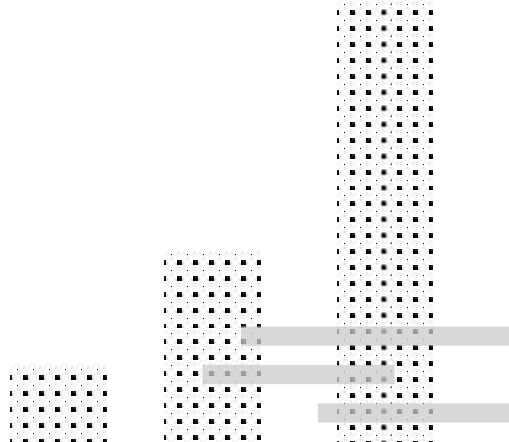
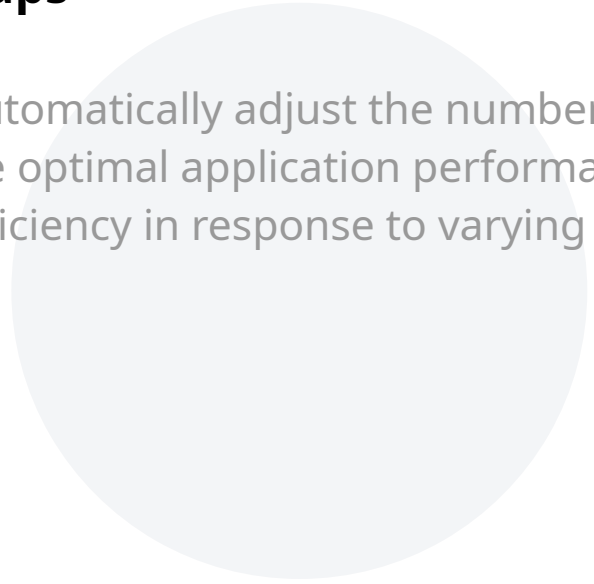


What Auto Scaling Groups are a feature of AWS that enable the automatic scaling of EC2 instances to meet changing application demands.

## **AWS Auto Scaling groups**



Why We use them to automatically adjust the number of EC2 instances to ensure optimal application performance, availability, and cost efficiency in response to varying demand



## The usual signals

CPU usage  
Memory usage  
Network bandwidth and allowance  
Disk bandwidth and IO  
Request count  
Instance or application health checks

## Less usual signals

Amount of available disk space  
Error rates  
Application specific metrics  
Business metrics  
OS metrics:

- available entropy
- file descriptor usage
- CPU context switches
- number of remaining ports

# Application / Business metrics

Socket backlog length

## Edge cases only

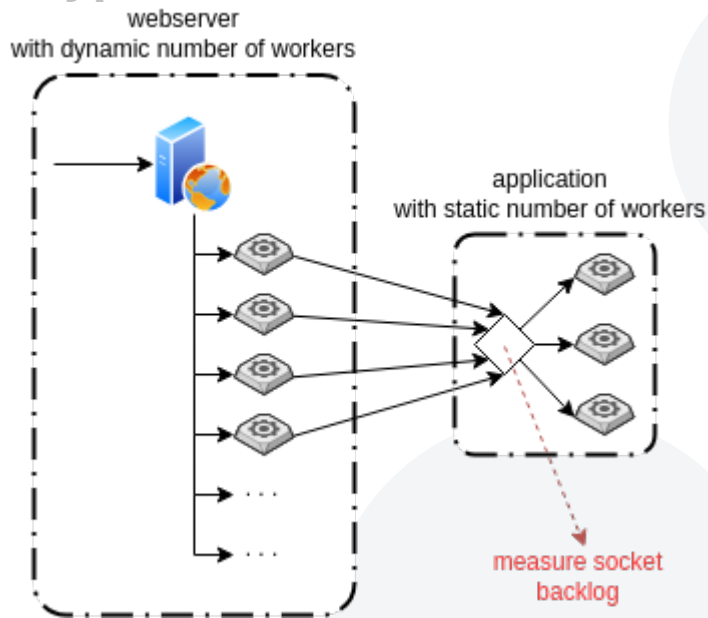
If your application can utilize all available EC2 resources, you should probably trigger scaling events on the resource metrics.

Measuring socket backlog can help you with scaling your instance groups, when requests to your application do not always cause high local resource usage and you want to ensure requests aren't queued for too long.

## Single threaded application

The significance of the backlog parameter is particularly important as the application's ability to handle incoming connection requests relies solely on this queue.

## Hypothetical environment



If you see your socket backlog length is increasing (\*over time), you could trigger a scaling event to reduce response latency.

**If you have the ability to fine-tune workers to consume all available local resources, that would be the preferred option.**

# Application / Business metrics

Socket backlog length

## POSIX

The POSIX standard specifies the behavior of the `listen()` function, including the backlog parameter. According to the standard, the backlog specifies the number of connections that can be queued before the system starts to refuse new connections. This allows the server to handle a certain number of connection requests concurrently, ensuring that a burst of connections does not immediately overwhelm the system.

The `listen()` function is defined as follows in the POSIX standard:

```
int listen(int socket, int backlog);
```

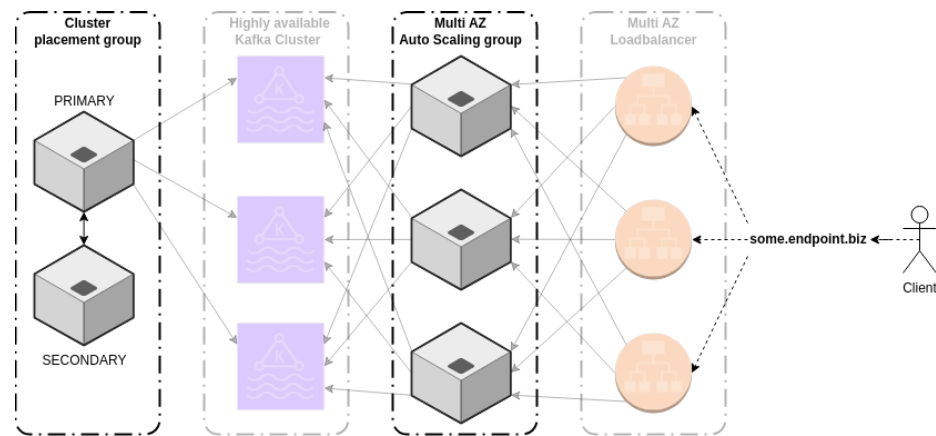
## Backlog

The maximum length to which the queue of pending connections for socket may grow. If a connection request arrives when the queue is full, the client may receive an error indicating that the server is busy.

Note: The actual maximum queue length may be influenced by the underlying operating system's implementation limits. Some systems might impose a hard limit on the backlog queue length, potentially capping the maximum value specified by the application.

`net.core.somaxconn`

What

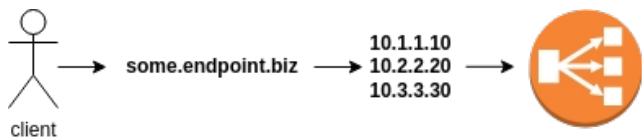


## Naive network path discovery

Why Sometimes latency matters.

## Naive path discovery

It's not a great idea, results aren't always precise, but it's cheap to implement on the client side



## Limitations

Server side must allow multiple connections from the same IP.

Server side must provide a timestamp for comparison.

Client side processing time may affect results.

## Caution

Large amount of concurrent connections may overwhelm the server side, causing a denial of service. Align with the service owner before attempting such approach on production systems.

# Naive path discovery

If the demo doesn't work

