

15-319 / 15-619
Cloud Computing

Overview 4

February 23rd, 2021

Administrative - OH & Piazza

- **Make use of office hours**
 - Clearly describe the problem, provide full context!
 - [Piazza Course Staff](#)
 - [Google calendar in ET](#)
 - [Google calendar in PT](#)
- **Suggestions for using Piazza**
 - Read Piazza Post Guidelines ([@6](#) and [@9](#)) before asking questions
 - Read Piazza questions & answers carefully to avoid duplicates
 - Name the subject properly so that others can find your post
 - Try to ask a **public** question if possible so others can also benefit
 - Don't ask a public question about a quiz question

Administrative - Cloud spending

- Monitor AWS expenses regularly
- Always do the cost calculation before launching services
- **Keep in mind that: 6-hour delay for AWS to update their logs on spending**
 - Accurate and timely expense reports are hard
 - The cost logs every 6 hours may be inaccurate
 - A resource may take days before it is reported in the logs
 - By the end of the billing cycle, the CSPs corrects the logs

Administrative - Cloud spending (cont.)

- Terminate your instances when not in use
- Stopped instances have EBS costs (\$0.1/GB-Month)
- Tag spot instances right after launch

Working within the specified budget is a very important skill to learn

Important - Compromised Accounts

- **DON'T EVER EXPOSE YOUR AWS CREDENTIALS!**
 - Github
 - Bitbucket
 - Anywhere public...
- **DON'T EVER EXPOSE YOUR GCP CREDENTIALS!**
- **DON'T EVER EXPOSE YOUR Azure CREDENTIALS!**
 - ApplicationId, ApplicationKey
 - StorageAccountKey, EndpointUrl

Reflection of last week

- Conceptual content on OLI
 - Quiz 3 (Modules 3, 4)
- Project theme - **Big data analytics**
 - **Inverted Index:** Built an inverted index with MapReduce using TDD
 - **Wiki Data Parallel Processing Analysis:** Use MapReduce to process **36GB** compressed / **128GB** uncompressed wiki data
 - MapReduce application to filter records and calculate aggregate daily pageviews
 - **Data Analytics:** Use Jupyter Notebooks and the pandas library to analyze the data and answer questions

This Week

- **Quiz 3 (OLI Modules 5, 6)**
 - Due on **Friday**, February 26th, 2021, 11:59PM ET
- **Project 2.1 and Reflection**
 - Due on **Sunday**, February 28th, 2021, 11:59PM ET
- **Project 1.2 Discussion**
 - Due on **Sunday**, February 28th, 2021, 11:59PM ET
- **P1.2 Code Review**
 - Due on **Wednesday**, March 3rd, 2021, 11:59PM ET
- **Primers released this week**
 - Code Review
 - Intro to Containers and Docker
 - Kubernetes and Container Orchestration

OLI Module 5 - Cloud Management

Cloud Software stack - enables provisioning, monitoring and metering of virtual user “resources” on top of the Cloud Service Provider’s (CSP) infrastructure.

- Cloud middleware
- Provisioning
- Metering
- Orchestration and automation
- Case Study: Openstack - Open-source cloud stack implementation

OLI Module 6 - Cloud Software Deployment Considerations

- Programming on the cloud
- Deploying applications on the cloud
 - Build fault-tolerant cloud services
 - Load balancing
 - Scaling resources
 - Dealing with tail latency
 - Economics for cloud applications

Code Review

- Code review is the systematic examination of source code.
 - Goal 1: *Expose you to code review - making sure code achieves its objective using a sound approach (readability, safe, etc)*
 - Goal 2: *Expose you to alternative approaches*
 - Goal 3: *Have you develop good coding habits and skills that will be useful for your careers*
- **For Project 1.2, completing code review is worth 5 points, and it will contribute toward the total grade of Project 1.2**

Project 2

Overview

Scaling and Elasticity with

- VMs
- Containers
- Functions

- **2.1 Scaling Virtual Machines**
 - Horizontal scaling in / out using AWS APIs
 - Load balancing, failure detection, and cost management on AWS
 - Infrastructure as Code (Terraform)
- **2.2 Scaling with Containers**
 - Building your own container-based microservices
 - Docker containers
 - Manage multiple Kubernetes Cluster
 - Multi Cloud deployments
- **2.3 Functions as a Service**
 - Develop event driven cloud functions
 - Deploy multiple functions to build a video processing pipeline

Project 2.1 Learning Objectives

- **Design** solutions and invoke cloud APIs to programmatically provision and deprovision cloud resources for a dynamic load
- **Configure** and deploy an Elastic Load Balancer and an Auto Scaling Group on AWS
- **Develop** solutions that monitor cloud resource metrics to manage cloud resources with the ability to deal with resource failure
- **Analyze** a workload pattern and develop elasticity policies to maintain the Quality of Service (QoS) of a web service

Project 2.1 Learning Objectives (cont.)

- **Account** for cost as a constraint when provisioning cloud resources and analyze the performance tradeoffs due to budget restrictions
- **Orchestrate** infrastructure on the cloud using Terraform as part of the deployment process

Overview of Quality of Service (QoS), Latency and Cloud Elasticity

- Quality of Service (QoS)
- Load patterns for web services
- Vertical scaling (Scale up/down)
- Horizontal scaling (Scale out/in)
- Load balancers
- Autoscaling groups
- Resource monitoring (CloudWatch)

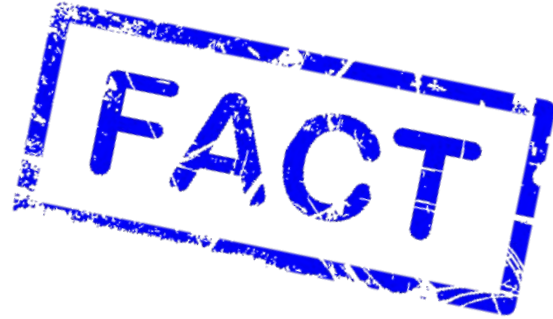
Quality of Service (QoS)

Quantitatively Measure QoS

- **Performance: Throughput, Latency**
(Very helpful in Project 2 & Team Project)
- **Availability:** the probability that a system is operational at a given time
(Project 2)
- **Reliability:** the probability that a system will produce the correct output

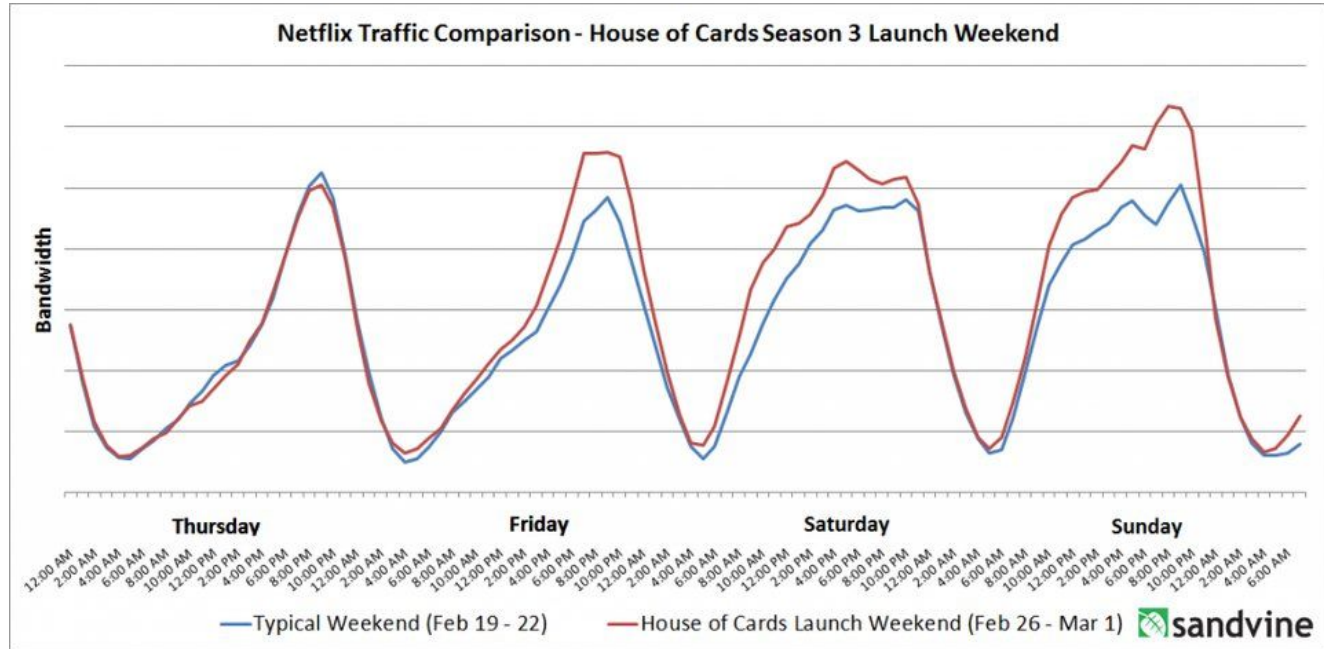
QoS Matters

- Amazon found every **100ms** of latency cost them **1%** in sales (~\$1B)



Reality, human patterns...

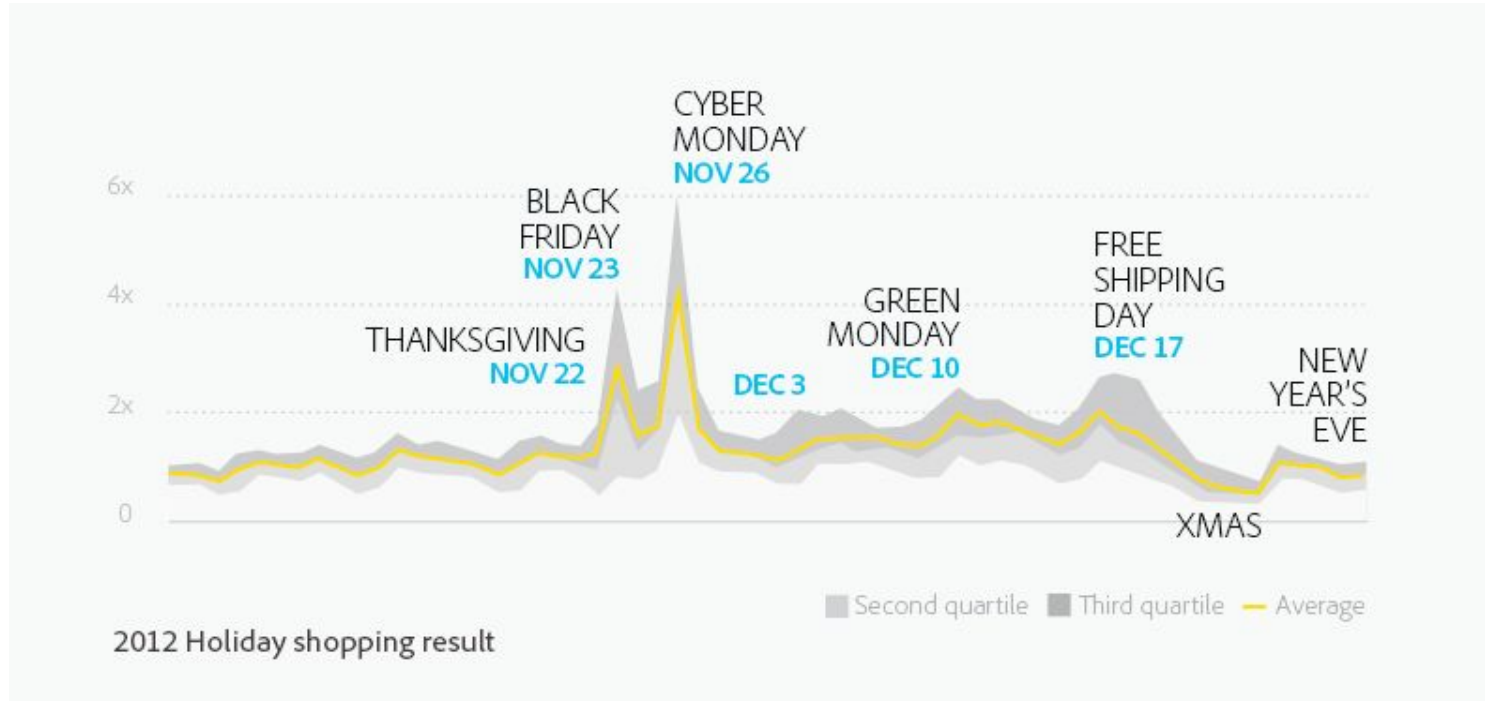
- Daily
- Weekly
- Monthly
- Yearly
- ...



The Ferenstein Wire

Reality, human patterns...

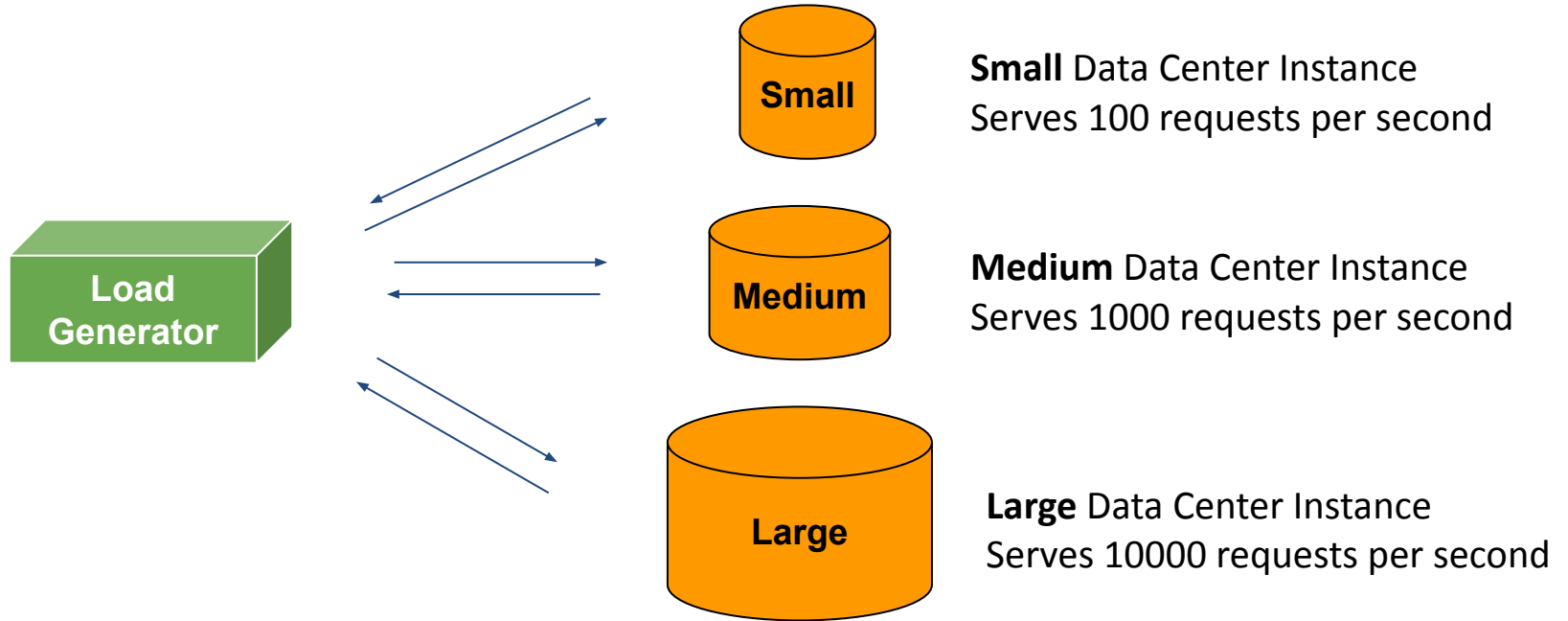
- Daily
- Weekly
- Monthly
- **Yearly**
- ...



Cloud Comes to the Rescue!

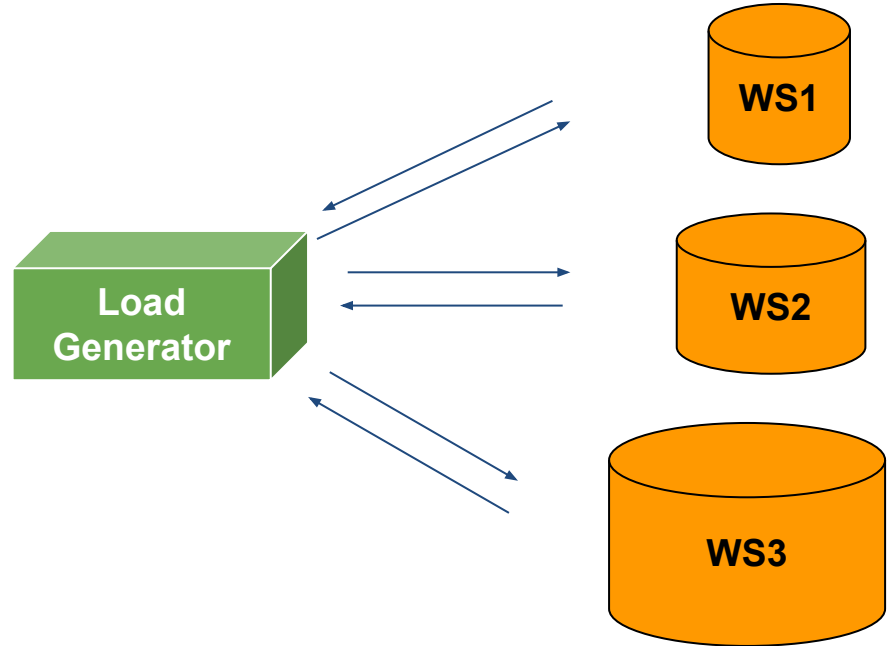
Scaling!

P0: Vertical Scaling

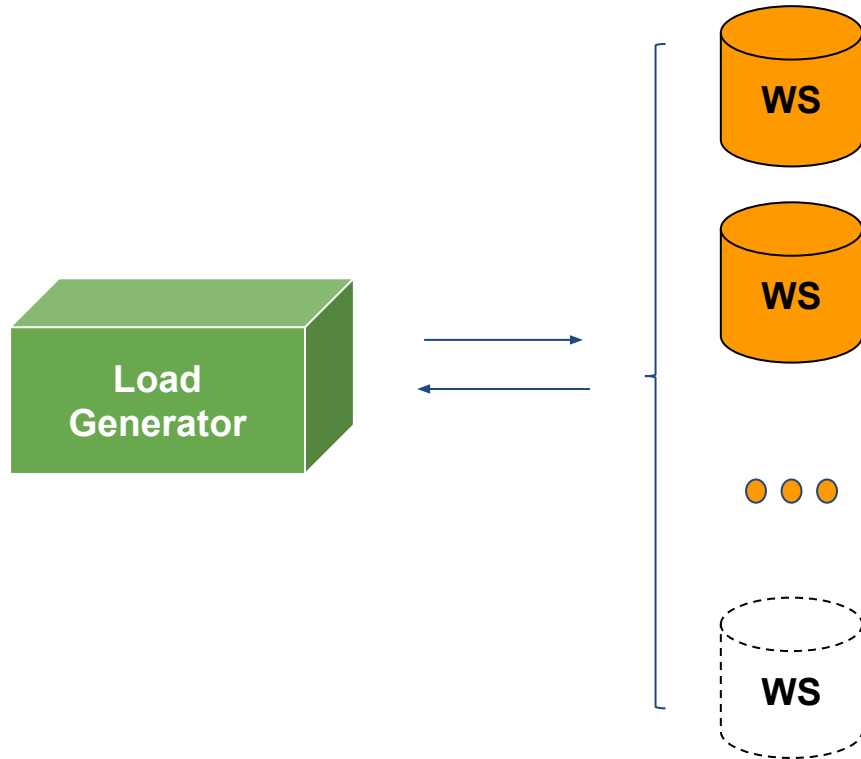


P0: Vertical Scaling Limitation

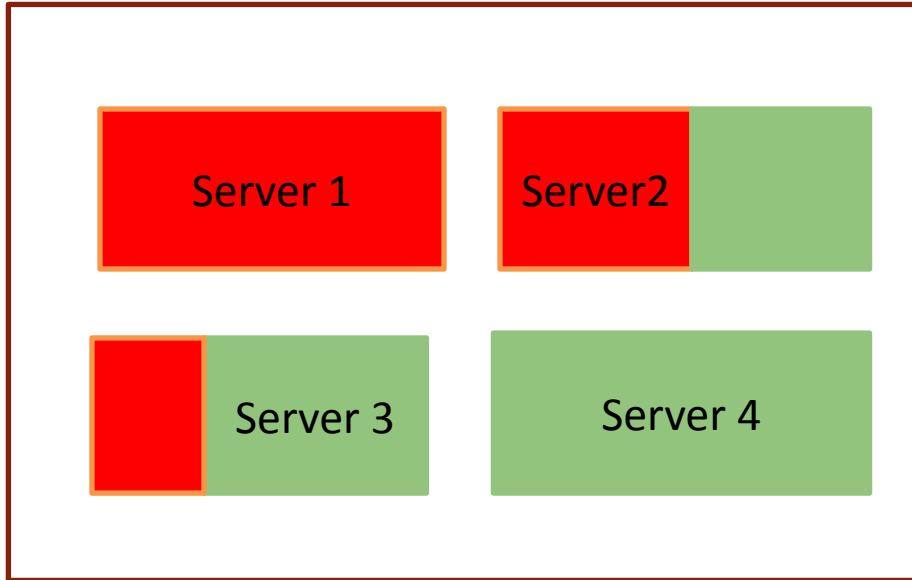
- However, one instance will always have limited resources
- Reboot/Downtime



Horizontal Scaling



How do we distribute load?

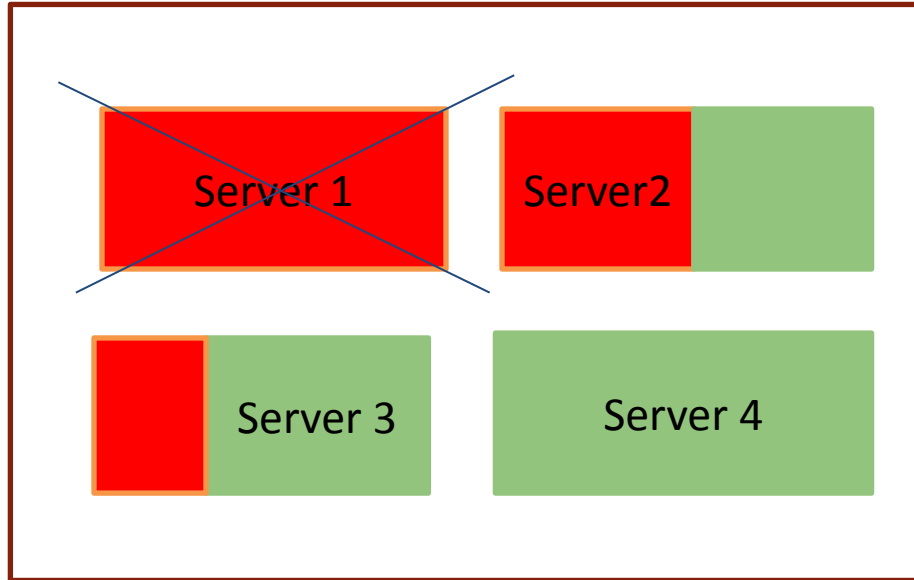


CPU utilization, memory utilization...



Available capacity

Instance Failure?



CPU utilization, memory utilization...

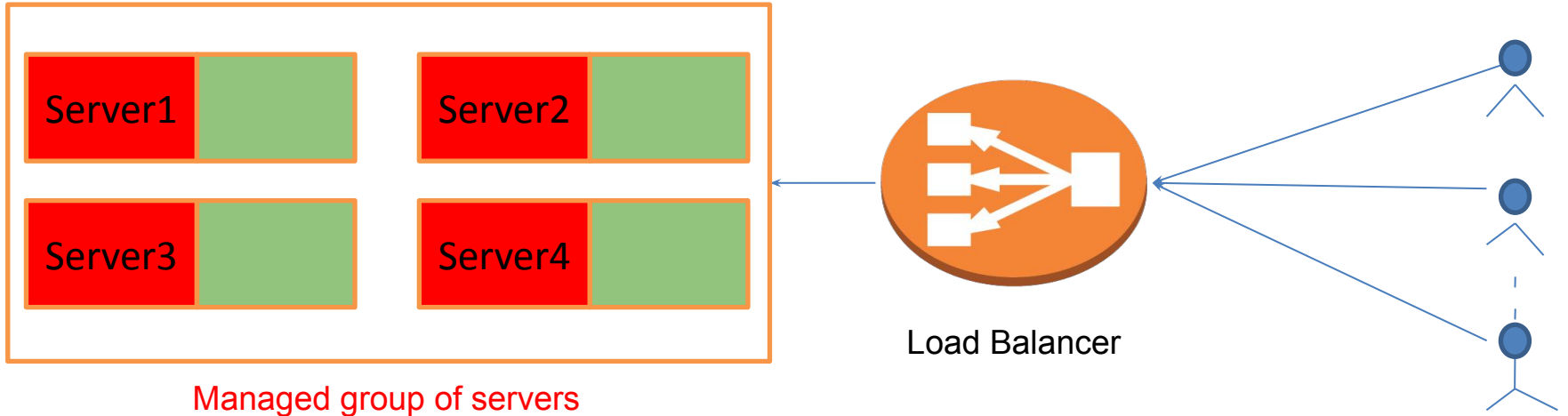


Available capacity

What You Need

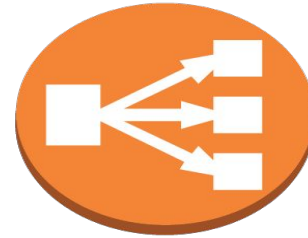
- Make sure that the workload is even on each server
- Do not assign a load to servers that are down
- Increase/Remove servers according to a changing load

How does a cloud service help solve these problems?



Load balancer

- “Evenly” distribute the load
- A simple distribution strategy
 - Round Robin
- Load check
- Health check



Load Balancer

- *What if the Load Balancer becomes the bottleneck?*
 - Elastic Load Balancer (ELB)
 - Could scale up based on load
 - Elastic, but it still takes time
 - Through the warm-up process

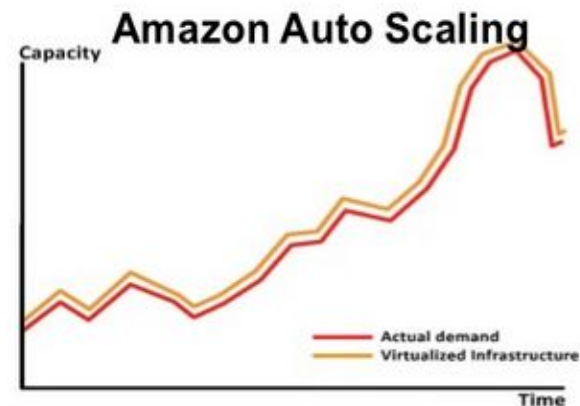
Scaling

Manual Scaling:

- Over provisioning and low utilization
- Expensive on manpower
- Lose customers

Autoscaling:

- Automatically adjust the size based on demand
- Flexible capacity and scaling sets
- Save cost



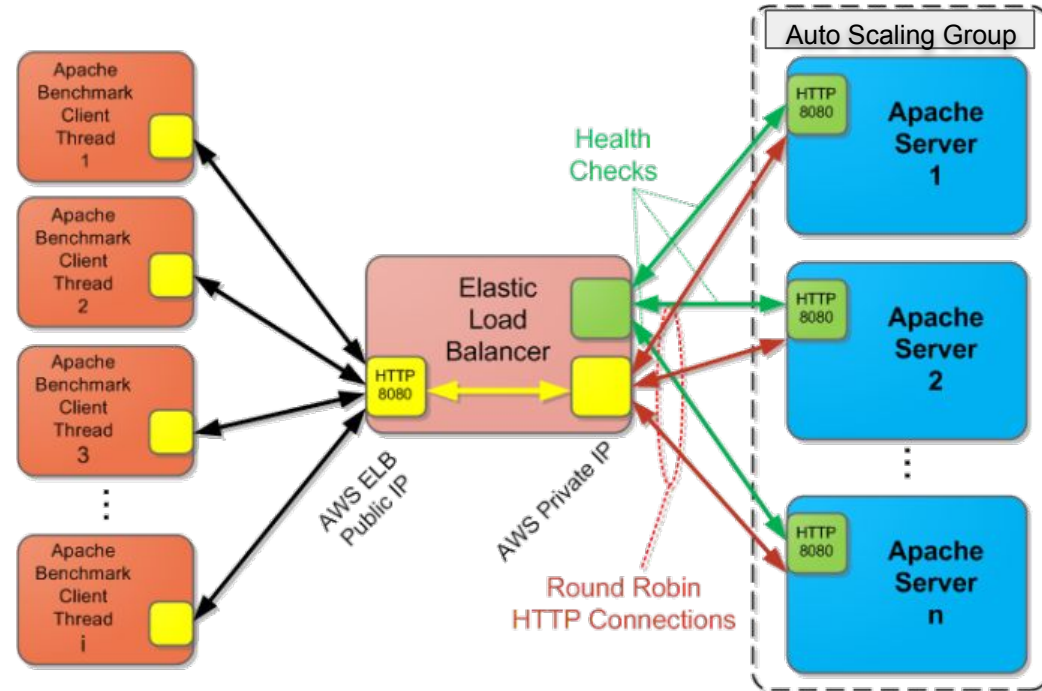
AWS Autoscaling

Auto Scaling on AWS

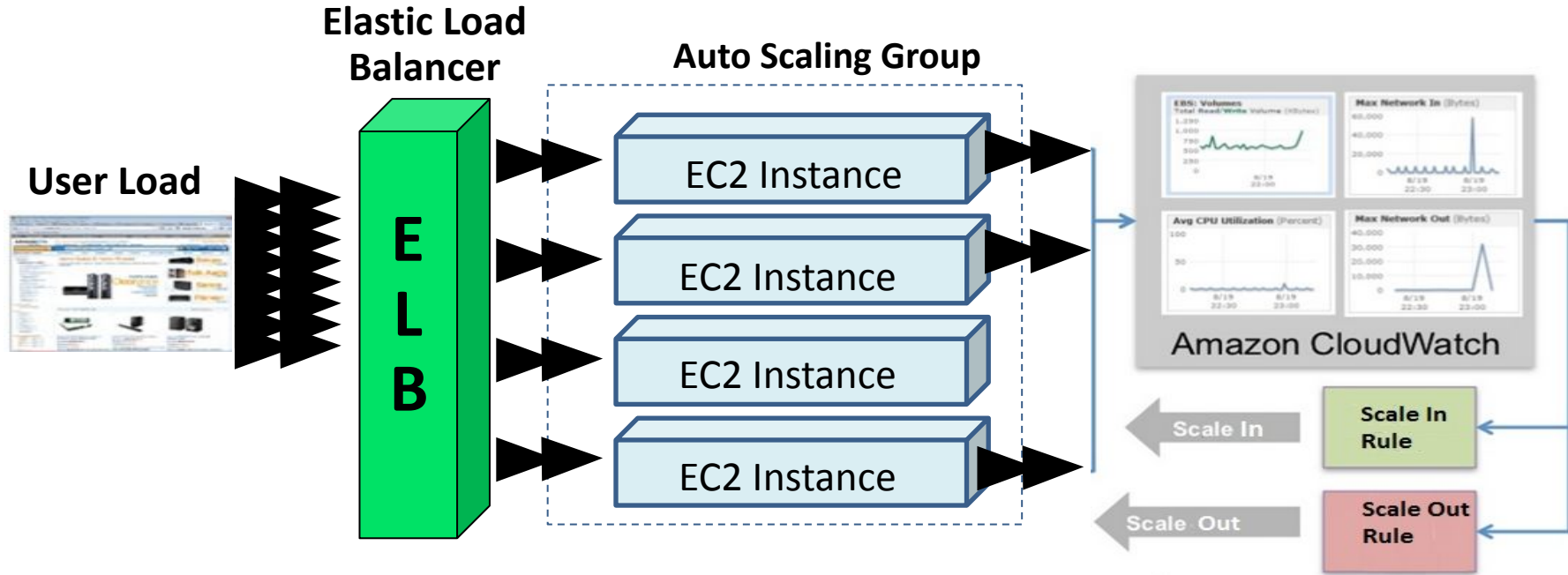
Using the AWS APIs:

- ELB
- Auto Scaling Group
- EC2
- CloudWatch
- Auto Scaling Policy

You can build a load balanced auto-scaled web service.



Amazon Auto Scaling Group



Amazon CloudWatch Alarm

CloudWatch: Application ELB ▾

1h 3h 12h 1d 3d 1w custom ▾

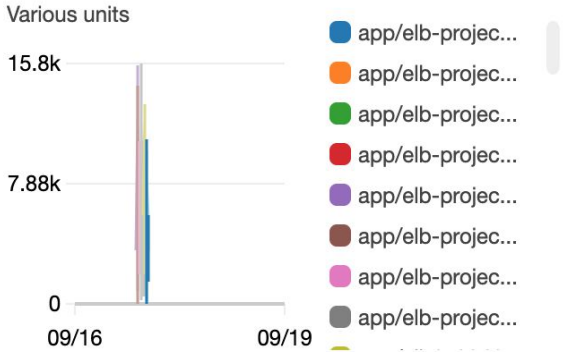
Actions ▾  ▾

All resources ▾

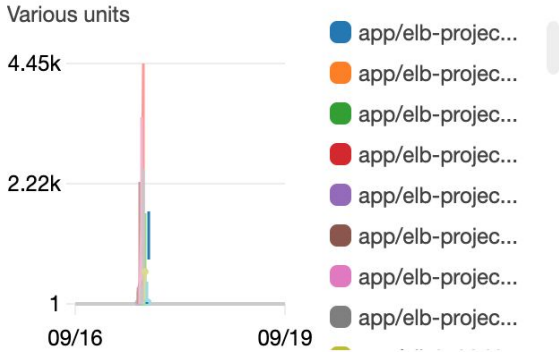
Service dashboard ▾

ALARM 0 INSUFFICIENT DATA 0 OK 0

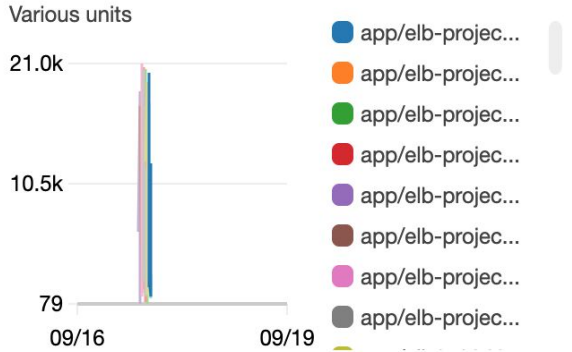
Request Count Sum



HTTP 5XX Count

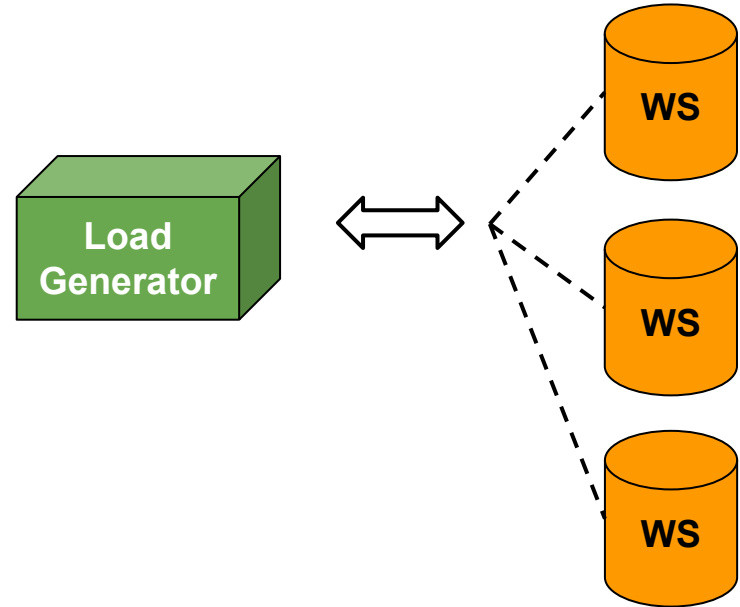


Active Connection Count Sum



Project 2.1 Scaling on AWS

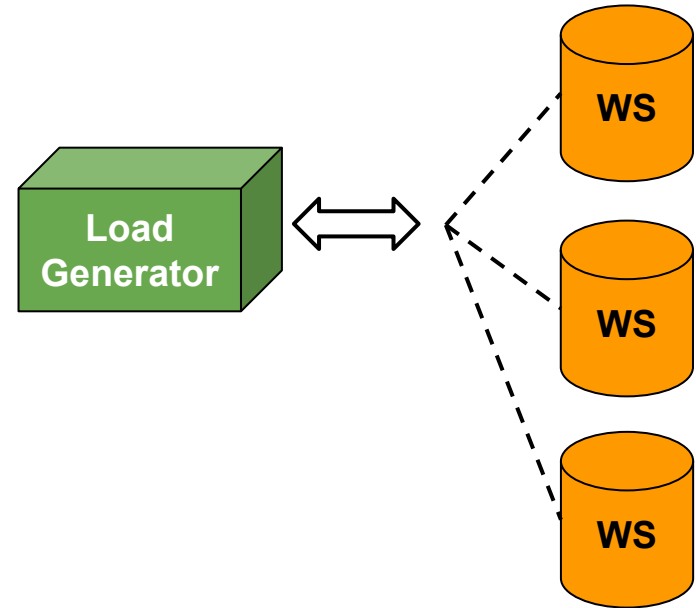
- **Task 1**
 - **AWS Horizontal Scaling**
- **Task 2**
 - **AWS Auto Scaling**
- **Task 3**
 - **AWS Auto Scaling with Terraform**



Project 2.1 Scaling on AWS

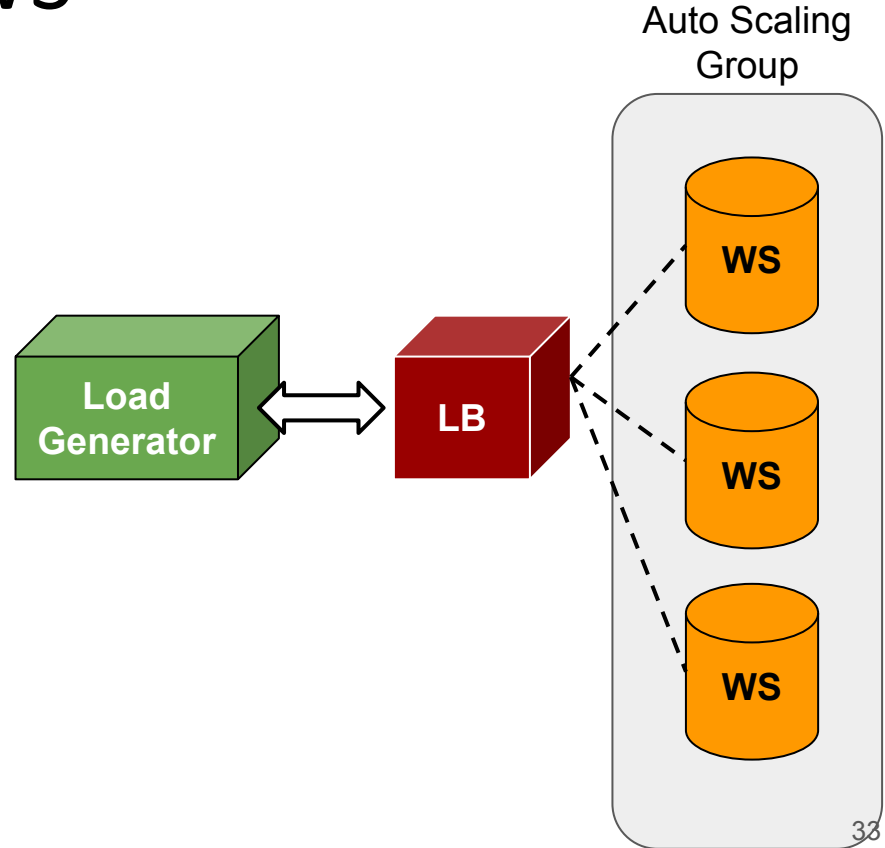
Task 1 - AWS Horizontal Scaling

- Implement Horizontal Scaling in AWS
- Write a program that launches web service instances and ensures that the target total RPS is reached
- Your program should be fully automated: launch LG → submit password → Launch WS → start test → parse log → add more WS...



Project 2.1 Scaling on AWS

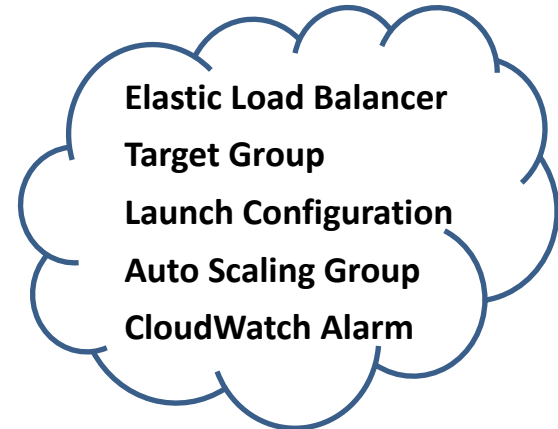
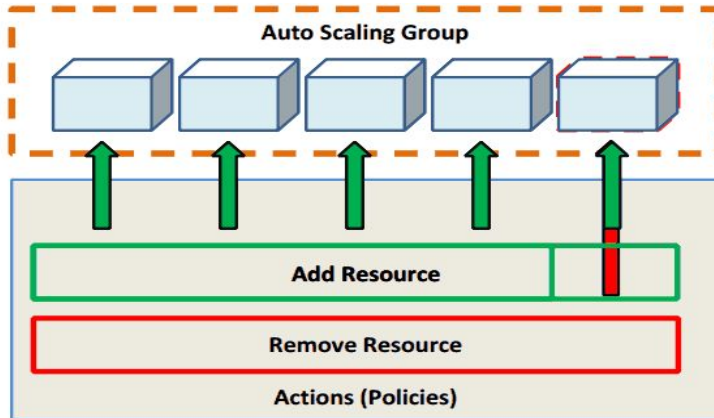
- Task 1
 - AWS Horizontal Scaling
- Task 2
 - AWS Auto Scaling
- Task 3
 - AWS Auto Scaling with Terraform



Project 2.1 Scaling on AWS

Task 2 - AWS Auto Scaling

- Programmatically create LG, Application Load Balancer (ALB), Auto-Scaling Group (ASG) with Auto Scaling Policy, launch configuration, and target group
- Adjust Scale-Out and Scale-In policies if necessary
- Your solution also needs to be fault tolerant
- Health configurations are important



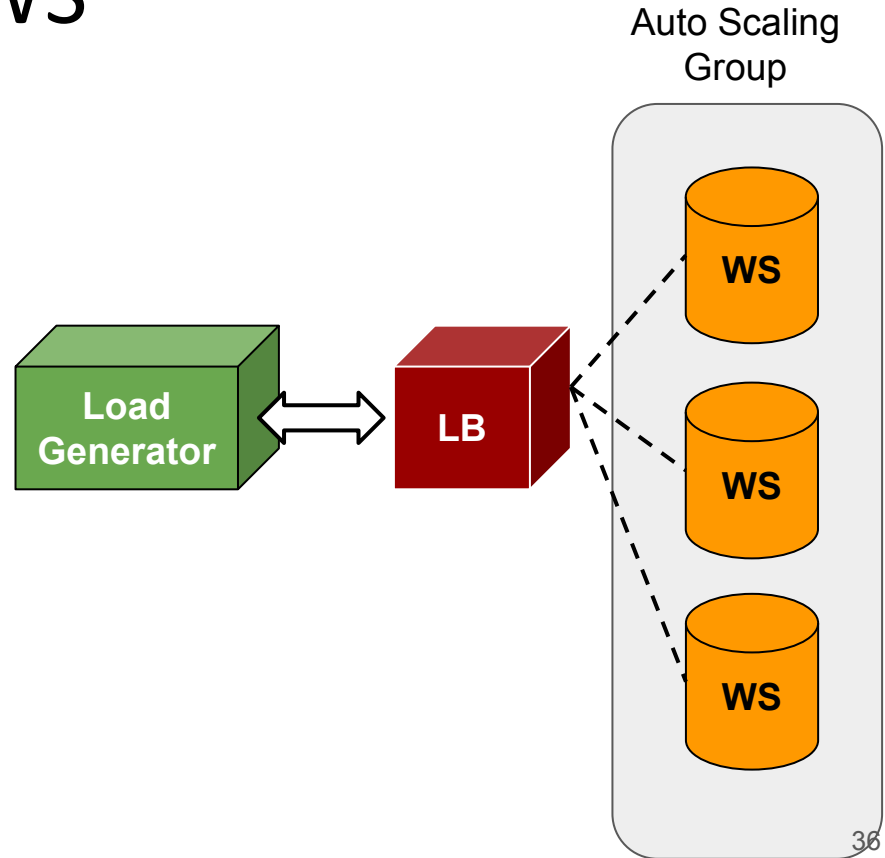
Hints for Project 2.1 AWS Autoscaling

Task 2 - AWS Auto Scaling

- Do a dry run via the console to make sure you understand the workflow
- The Autoscaling test could be very expensive!
 - On-demand, charged by per second, do not blindly launch tests
- CloudWatch monitoring is helpful for policy tuning
- Observe and analyze the pattern, experiment with a policy, collect data to verify why it achieved a certain performance, and iterate until you achieve your goal
- Explore ways to check if your instance is ready
- **You will need spend a lot of time understanding the API documents**

Project 2.1 Scaling on AWS

- Task 1
 - AWS Horizontal Scaling
- Task 2
 - AWS Auto Scaling
- **Task 3**
 - **AWS Auto Scaling with Terraform**



Project 2.1 Scaling on AWS

Task 3 - AWS Auto Scaling with Terraform:

- Read the “Infrastructure as Code” primer to learn about infrastructure automation
- Make sure that **terraform plan** generates the expected resource
- Make sure that all the variables (AMI ID, CloudWatch thresholds, Security Group names, etc.) are manually specified in the terraform main file

Project 2.1 Code Submission

- To submit each task, run the submitter within the individual tasks folder. The submitter executes your code for that task and submits your code to TheProject.Zone
- We will grade your code for each task **separately**.
- There are a lot of manual scoring points, follow our directions and don't take any shortcuts

Penalties for Project 2.1

Violation	Penalty of the project grade
Spending more than \$20 for this project phase on AWS	-10%
Spending more than \$40 for this project phase on AWS	-100%
Failing to tag all your resources in either parts (EC2 instances, ELB, ASG) for this project with the tag: key=Project, value=2.1	-10%
Submitting your AWS/Andrew credentials in your code for grading	-100%
Using instances other than t3.micro (testing only) or m5.large for Horizontal scaling on AWS	-100%
Using instances other than t3.micro (testing only), m5.large for Autoscaling on AWS	-100%
Submitting executables (.jar, .pyc, etc.) instead of human-readable code (.py,.java, .sh, etc.)	-100%

Penalties for Project 2.1 (cont.)

Violation	Penalty of the project grade
Attempting to hack/tamper the autograder in any way	-200%
Cheating, plagiarism or unauthorized assistance (please refer to the university policy on academic integrity and our syllabus)	-200%

AWS Cloud APIs



- AWS CLI ([link](#))
- AWS Java SDK ([link](#))
- AWS Python SDK ([link](#))

This Week Reminders

- **Quiz 3 (OLI Modules 5 & 6)**
 - Due on **Friday**, February 26th, 2021, 11:59PM ET
- **Project 2.1 and Reflection**
 - Due on **Sunday**, February 28th, 2021, 11:59PM ET
- **Project 1.2 Discussion**
 - Due on **Sunday**, February 28th, 2021, 11:59PM ET
- **P1.2 Code Review**
 - Due on **Wednesday**, March 3rd, 2021, 11:59PM ET
- **Primers released this week**
 - Code Review
 - Intro to Containers and Docker
 - Kubernetes and Container Orchestration