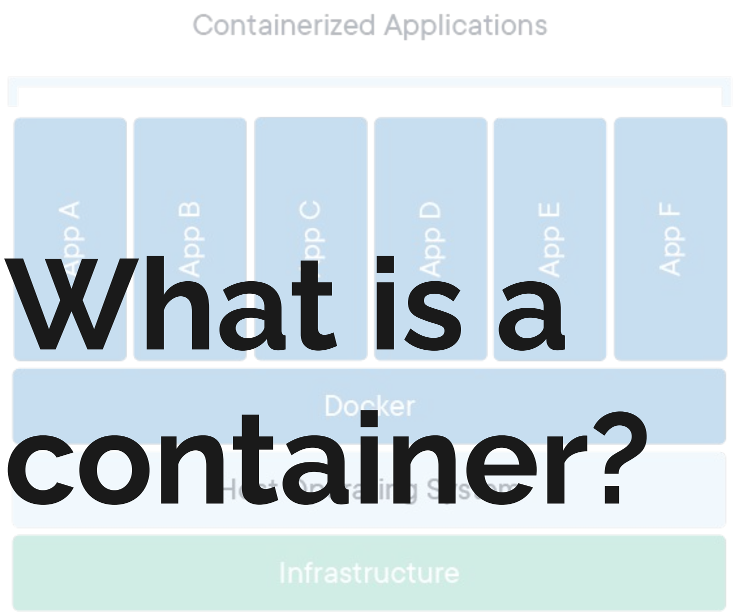
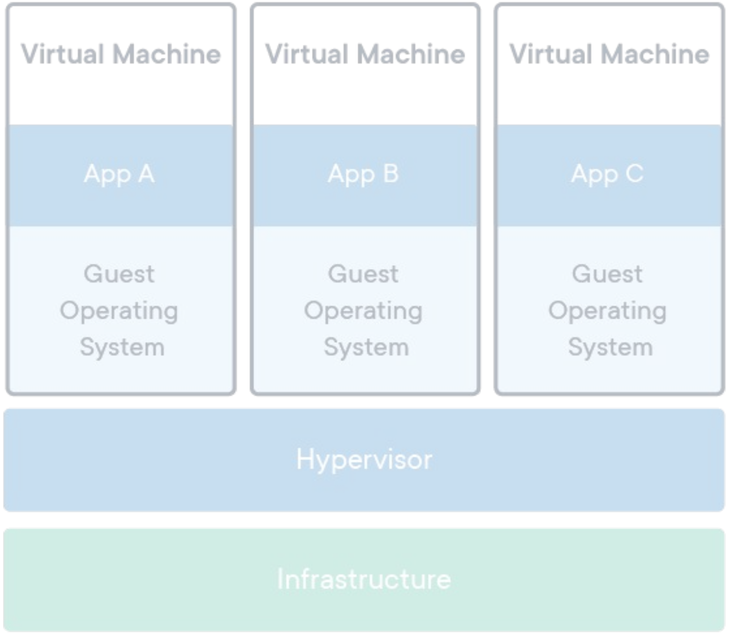
Prof. Emiliano Casalicchio - Pietro Costanzi Fantini 1982805 | Cristian Apostol 2002291

Kubernetes Orchestration on AWS

Cloud Computing - September 2025

What is a container?





Containers and VMs

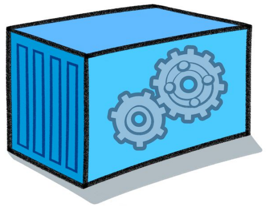
01

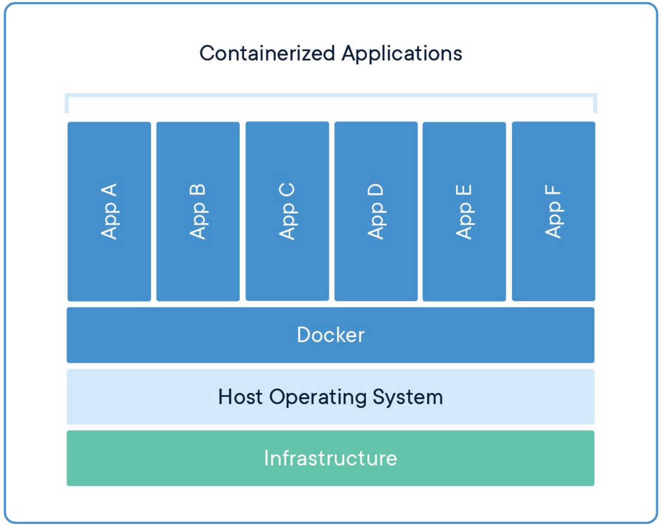
Containers are **isolated environments** that run as an isolated process in userspace on the host OS, sharing the kernel with other containers.

They are defined by a simple file that lists the various **dependencies** and **applications** each container needs to have in order to operate.

They are what we call a **operating system virtualization**.

They require a container engine to run, such as the **Docker Engine**.



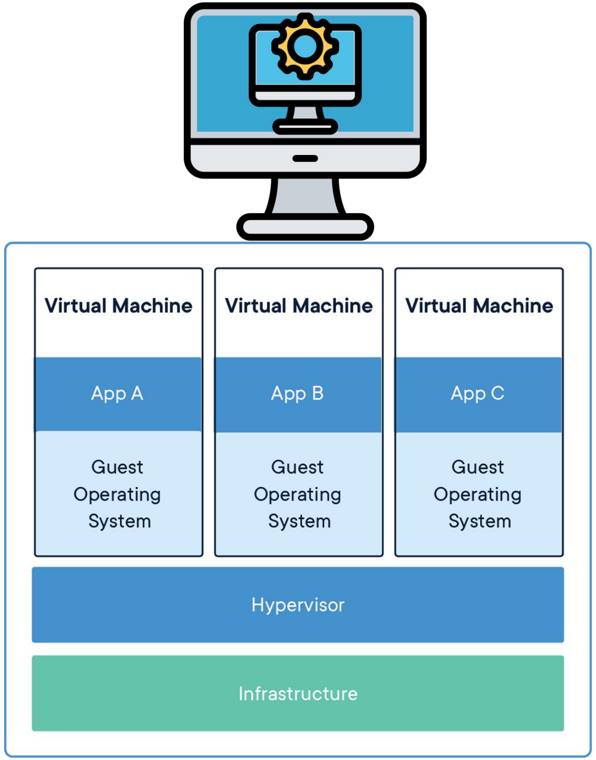


Containers and VMs

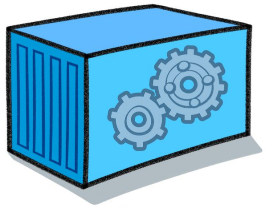
Virtual machines are isolated environments that can either be placed bare-metal (**TYPE-I Hypervisor**) or on top of a host OS (**TYPE-II Hypervisor**).

They are **isolated environments** that have their own guest OS.

There are levels to the virtualization which affect the efficiency of the virtual machine (**Full-virtualization, Paravirtualization**).



Containers and VMs



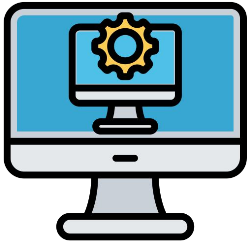
PROS

★ Lightweight and fast
★ Highly efficient
★ Consistent
★ Microservices Friendly

CONS

❖ Weaker isolation
❖ OS dependency

Containers and VMs



PROS	CONS
★ Strong isolation	❖ High overhead
★ Great flexibility	❖ Slower and larger
★ Highly portable	❖ Less efficient
★ Full HW abstraction	

What is Kubernetes?

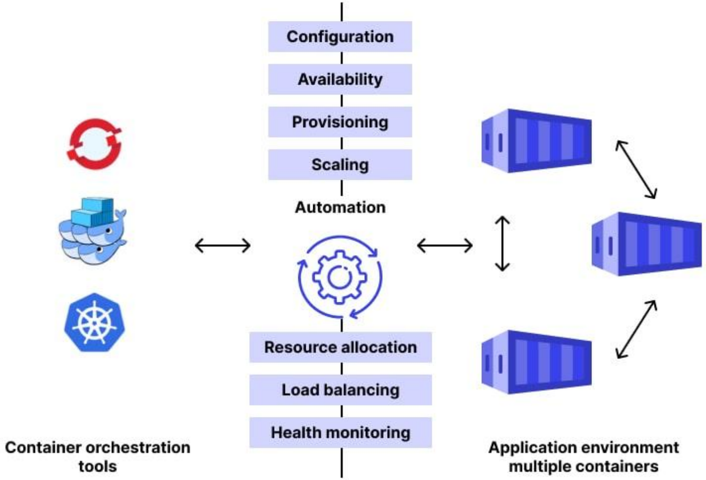
Kubernetes

01

Orchestration is the concept of **automating** a series of individual tasks to work together as a process or workflow.

Container Orchestration allows cloud and application providers to define how to select, deploy, monitor, and dynamically control the configuration of multi-container packaged applications in the cloud.

Main features include: **resource limit control, scheduling, load balancing, health check, fault tolerance, and auto-scaling.**



Kubernetes

Kubernetes is a **portable**, **extensible**, **open source** platform for managing containerized workloads and services, that facilitates both declarative configuration and automation.

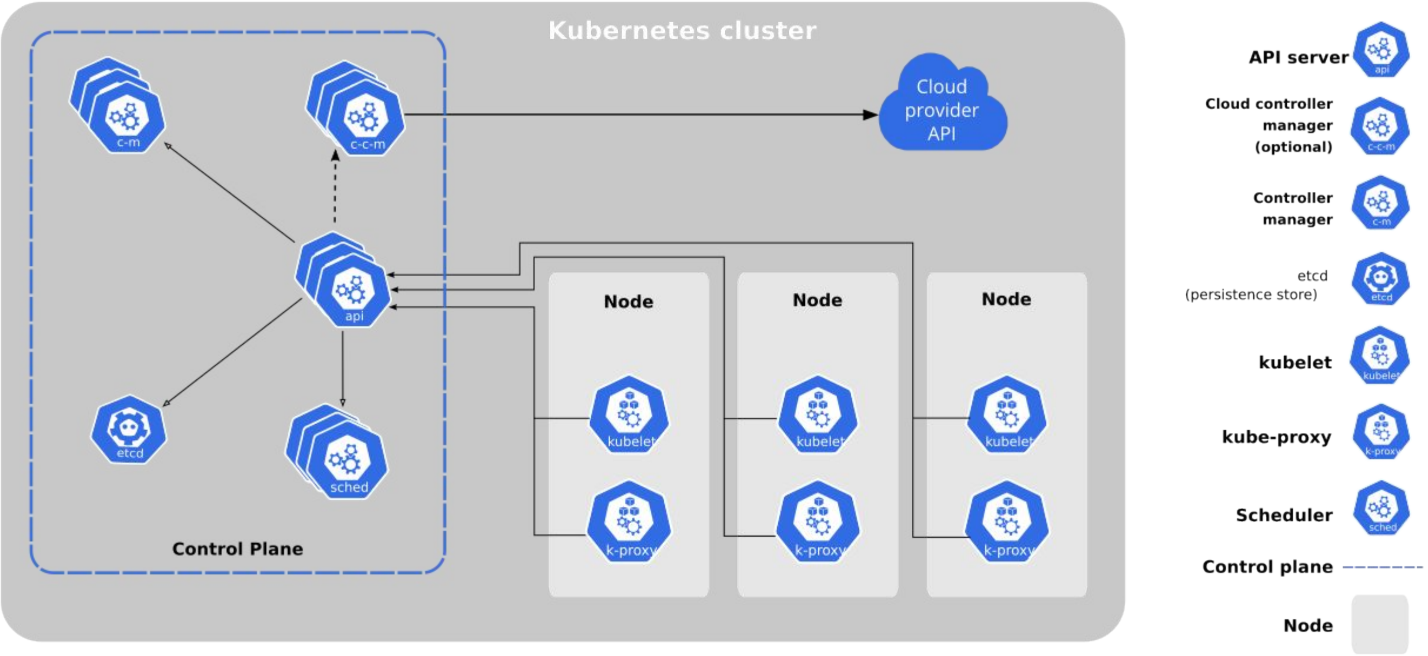
It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.

Kubernetes

- ★ Service discovery and load balancing
- ★ Storage orchestration
- ★ Automated rollouts and rollbacks
- ★ Automatic bin packing
- ★ Self-healing
- ★ Secret and configuration management
- ★ Batch execution
- ★ Horizontal scaling
- ★ IPv4/IPv6 dual-stack
- ★ Designed for extensibility

kubernetes

Kubernetes


Kubernetes cluster
c-m
c-c-m
Cloud provider API
api
etcd
sched
Control Plane
Node
Node
Node
kubelet
kubelet
kubelet
k-proxy
k-proxy
k-proxy
API server
api
Cloud controller manager (optional)
c-c-m
Controller manager
c-m
etcd (persistence store)
etcd
kubelet
kubelet
kube-proxy
k-proxy
Scheduler
sched
Control plane
Node

Prof. Emiliano Casalicchio - Pietro Costanzi Fantini 1982805 | Cristian Apostol 2002291

Project Overview

Project Overview

03

Goal

To build, test, and evaluate a scalable web application on a self-managed Kubernetes cluster deployed on AWS.

Core Focus

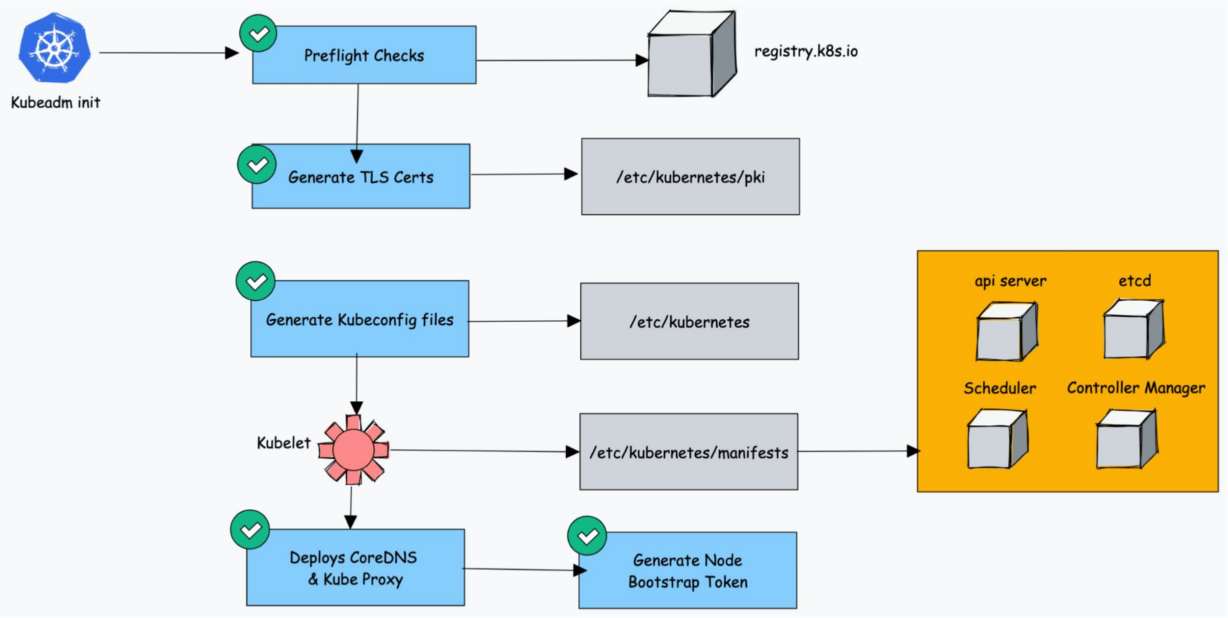
Demonstrate the effectiveness and behavior of the Kubernetes Horizontal Pod Autoscaler (HPA) in response to various user traffic patterns.

System Architecture

Kubernetes Cluster:

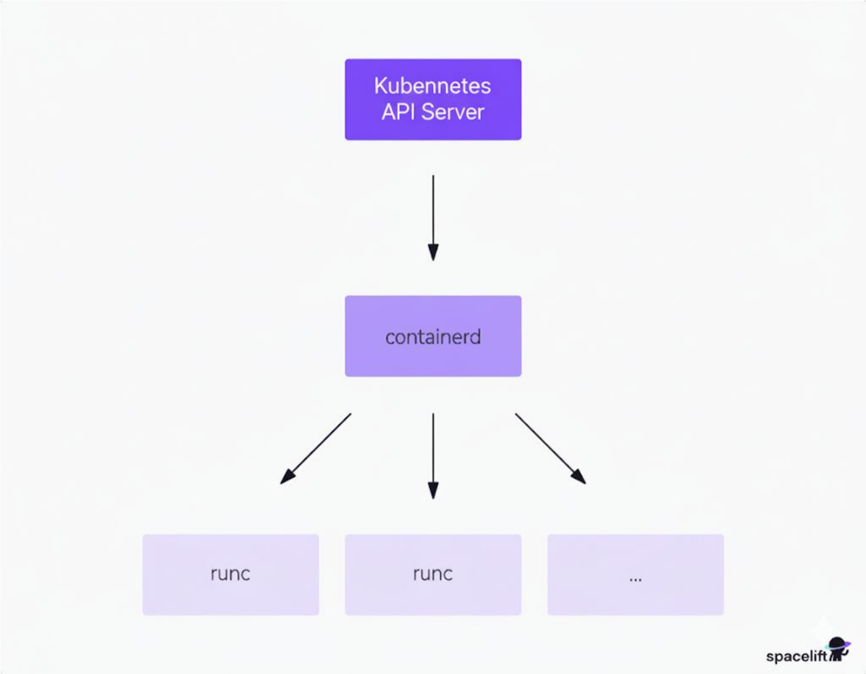
- **Setup**: Self-managed using **kubeadm** on AWS EC2 instances (**t2.medium - 2 CPU cores, 4 GB or RAM**).
- **Structure**: **1 Master Node** (Control Plane) and **4 Worker Nodes**.
- **Container runtime:** we used **containerd** in order to deploy a lightweight low-level container runtime.
- **Networking**: **Flannel** was used as the Container Network Interface (**CNI**) for pod-to-pod communication.

System Architecture



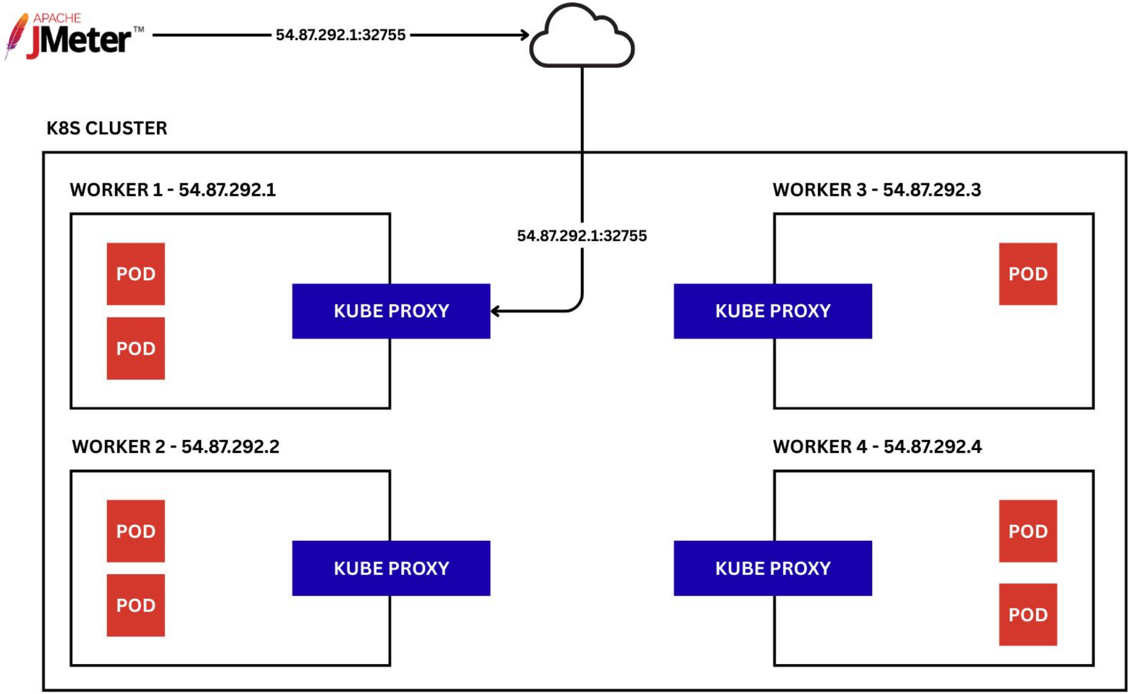
Kubeadm settings

System Architecture



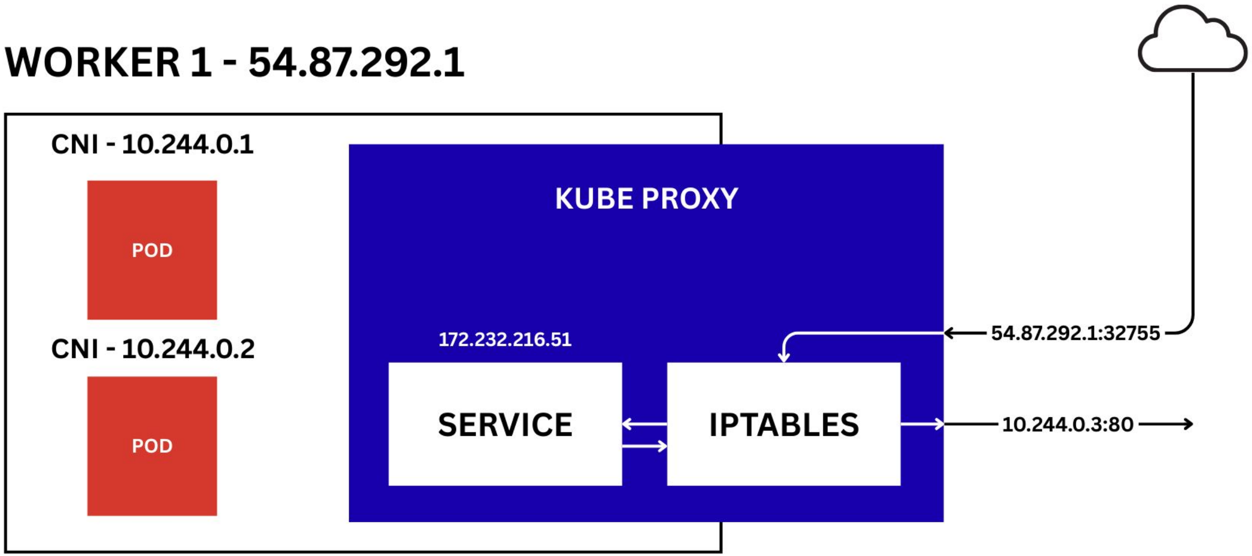
Kubeadm settings

System Architecture



Kubernetes cluster

System Architecture



Kubernetes cluster

System Architecture

Instances (5) Info

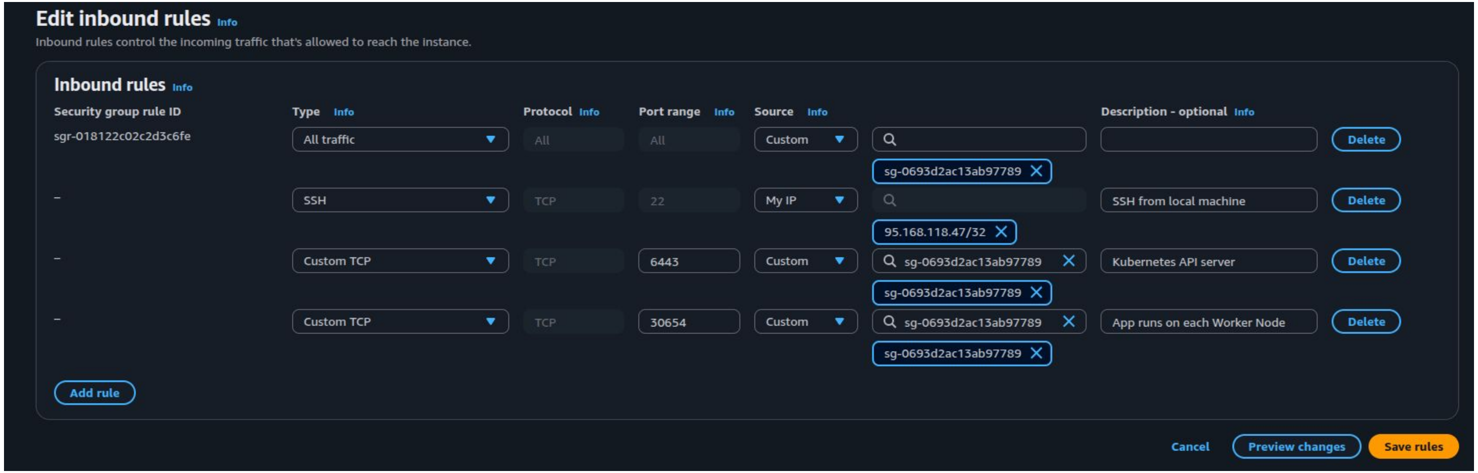
Connect | Instance state ▼ | Actions ▼ | Launch instances ▼

Find Instance by attribute or tag (case-sensitive) | All states ▼ | 1

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IF
k8s - master	i-091a4067c284bb0db	Running	t2.medium	2/2 checks passec	View alarms +	us-east-1b	ec2-54-234-148-48.co...	54.234.148.48	–
k8s - worker1	i-0e6b8dfbba07ee99a	Running	t2.medium	2/2 checks passec	View alarms +	us-east-1b	ec2-23-22-181-23.com...	23.22.181.23	–
k8s - worker2	i-051e8ef3b887489db	Running	t2.medium	2/2 checks passec	View alarms +	us-east-1b	ec2-13-220-55-54.com...	13.220.55.54	–
k8s - worker3	i-03a165495d909e5ee	Running	t2.medium	2/2 checks passec	View alarms +	us-east-1b	ec2-34-235-140-199.co...	34.235.140.199	–
k8s - worker4	i-09f89a17a71397ec0	Running	t2.medium	2/2 checks passec	View alarms +	us-east-1b	ec2-54-198-141-149.co...	54.198.141.149	–

Kubernetes cluster

System Architecture



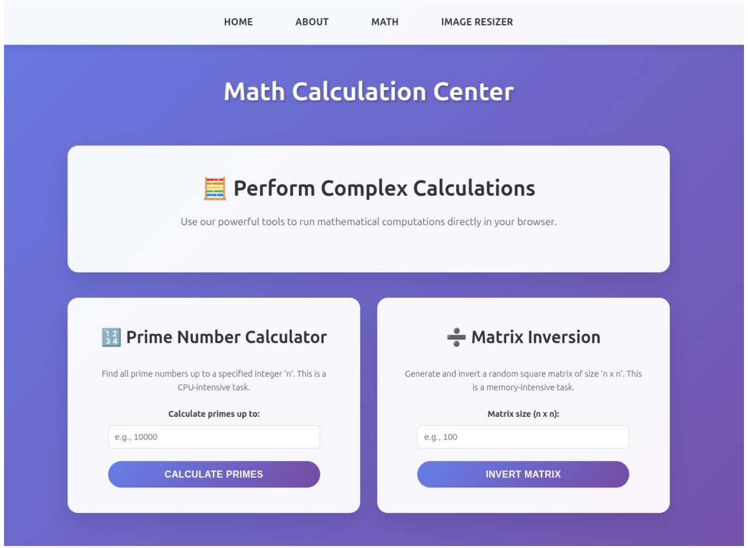
AWS Security Groups

System Architecture

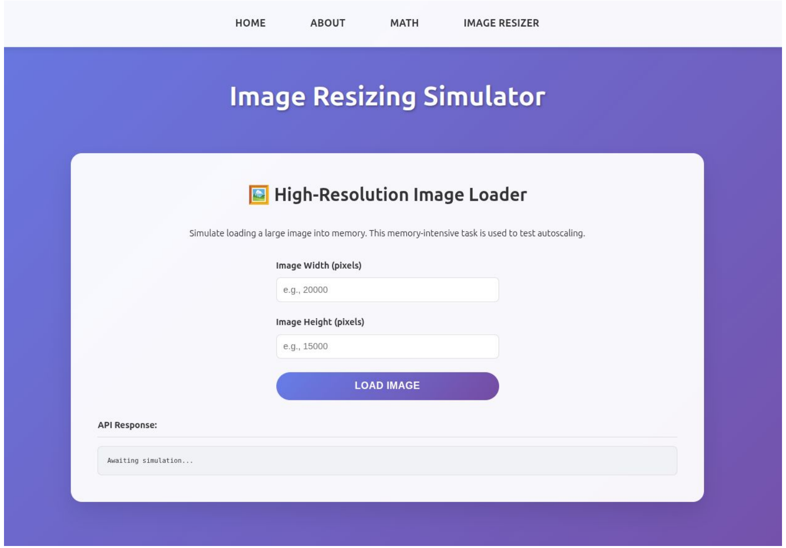
Application:

- **Framework**: A simple Python **Flask** web application.
- **Containerization**: Packaged into a **Docker image** and stored on **Docker Hub** for easy deployment.
- **Endpoints**:
  - **CPU-Intensive**: Prime number calculation and Matrix inversion.
  - **Memory-Intensive**: Image resizing simulator.

System Architecture



CPU-intensive endpoint



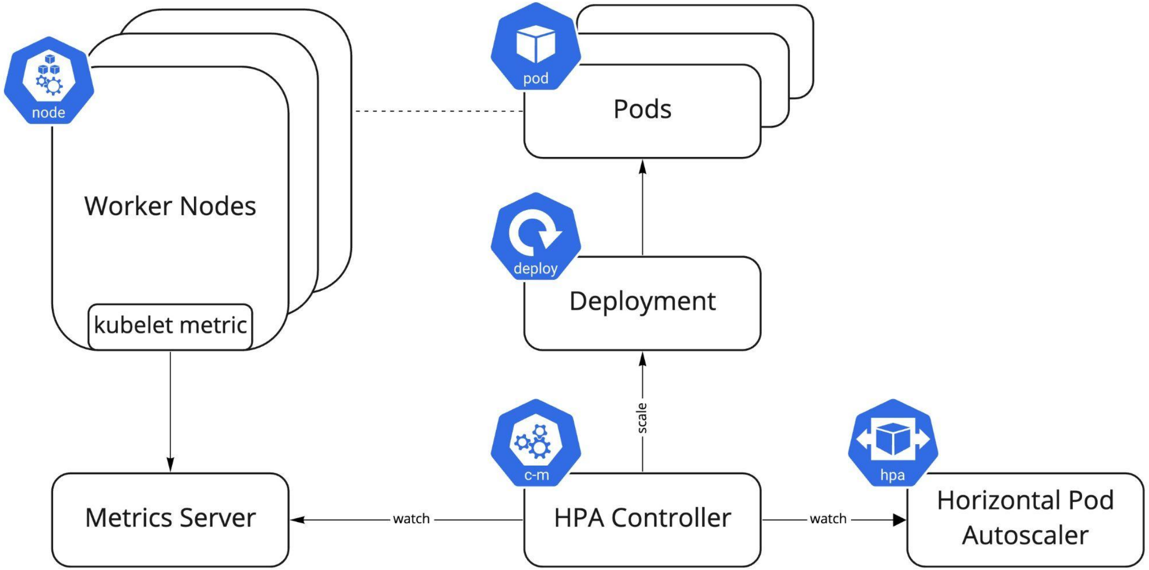
Memory-intensive endpoint

Autoscaling

Autoscaling is the process of automatically adjusting the number of running application instances (**pods**) in response to real-time workload demand.

In **Kubernetes**, the Horizontal Pod Autoscaler (**HPA**) monitors metrics such as CPU or memory utilization and scales the number of pods in **Kubernetes Objects**.

Autoscaling



Autoscaling

$$\text{desiredReplicas} = \left\lceil \text{currentReplicas} \times \frac{\text{currentMetricValue}}{\text{desiredMetricValue}} \right\rceil$$

System Evaluation

System Evaluation

04

We used a testing tool called **Apache JMeter**, chosen for its ability to simulate complex user scenarios and generate detailed performance reports.

We setup **2 key experiments**:

- **CPU-Intensive workload**: This experiment targeted the prime number calculation task endpoint (*/api/calculate primes*) and the matrix inversion task endpoint (*/api/invert matrix*) to trigger the CPU-based scaling policy of the HPA.
- **Memory-Intensive workload**: This experiment targeted the image-resizing endpoint (*/api/image/load*) in order to trigger the Memory-based scaling policy of the HPA.

System Evaluation

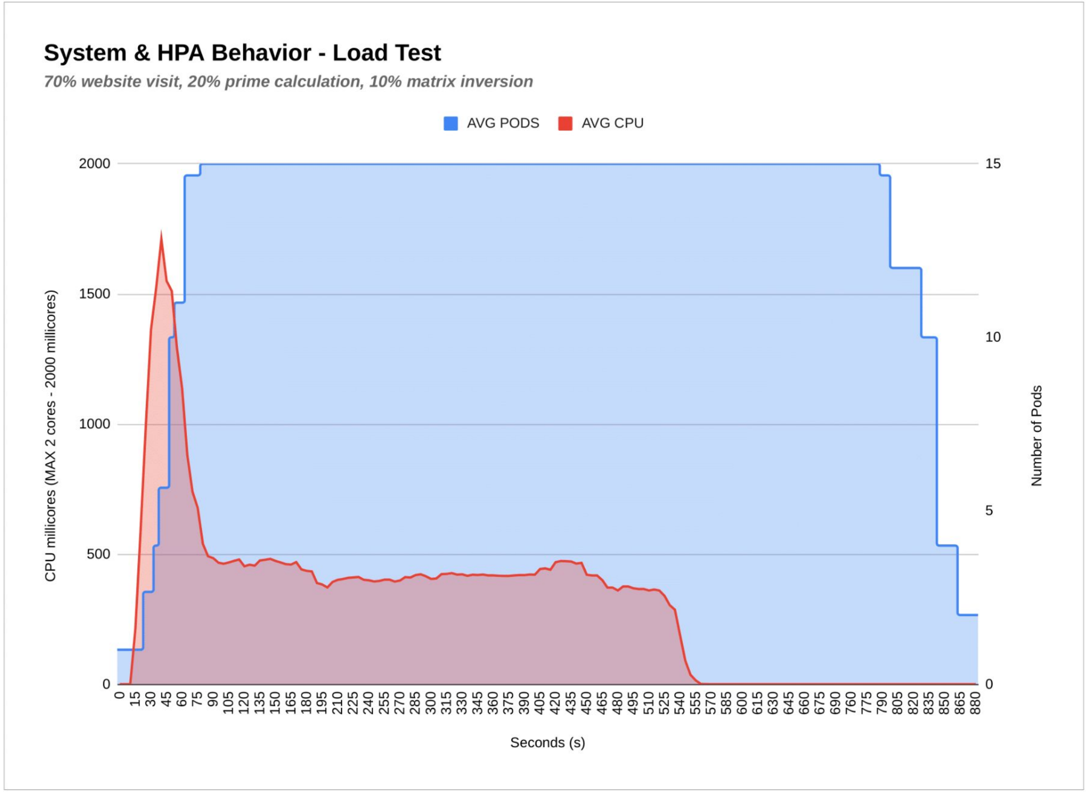
We setup **4 key test scenarios** to be performed and **averaged over 5 independent runs**:

- **Load Test**: Simulate normal, busy operating conditions.
- **Stress Test**: Intentionally push the system to its breaking point to find its limits.
- **Endurance Test**: Run a sustained load over a long period to check for stability and memory leaks.
- **Spike Test**: Evaluate the system's reaction time and elasticity to sudden bursts of traffic.

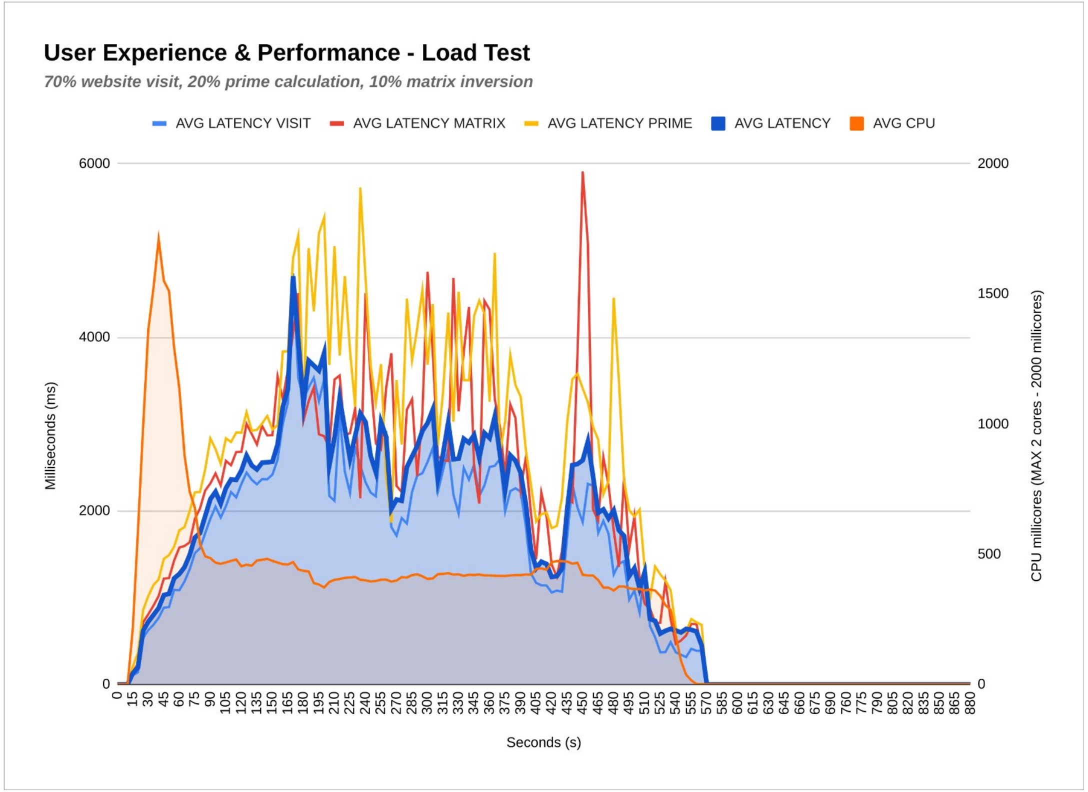
Prof. Emiliano Casalicchio - Pietro Costanzi Fantini 1982805 | Cristian Apostol 2002291

Experiment 1
CPU-Intensive

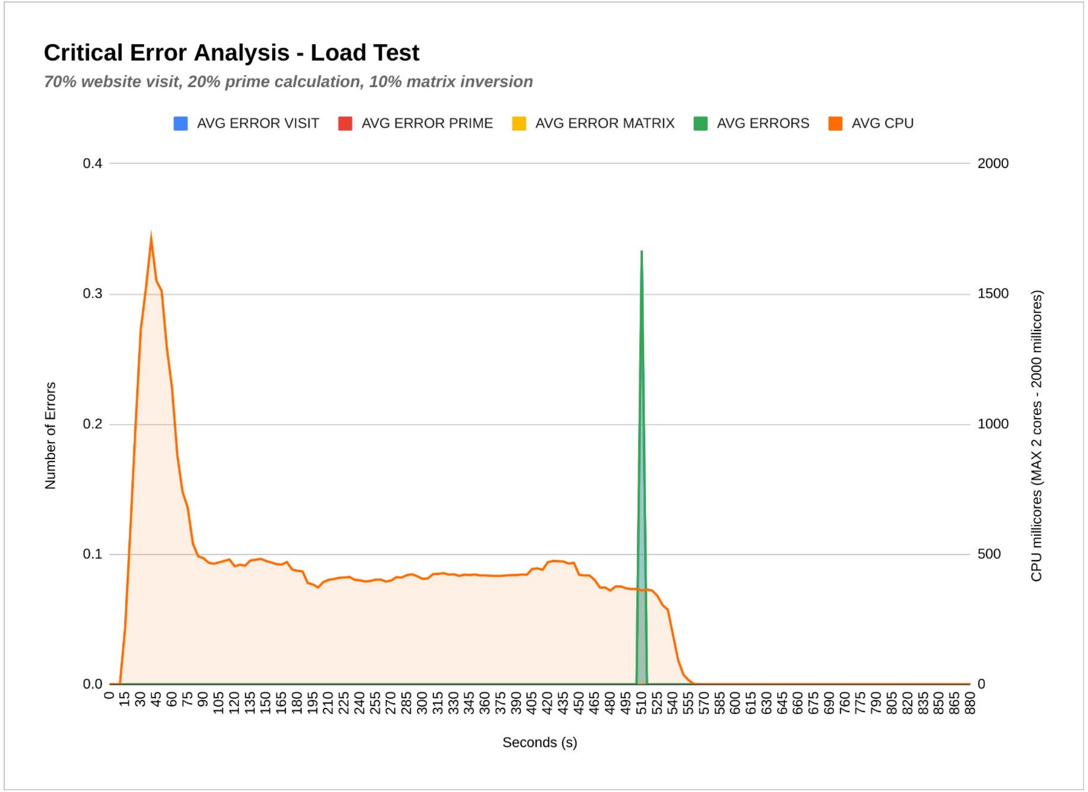
Experiment 1 - Load Test



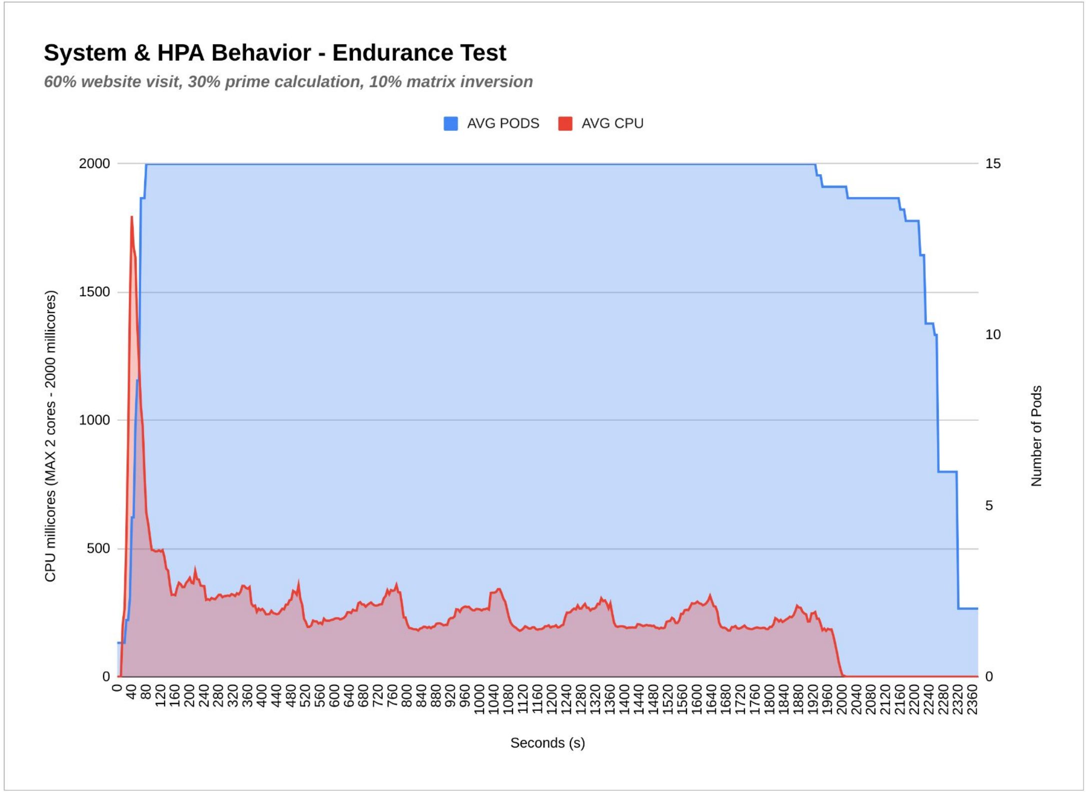
Experiment 1 - Load Test



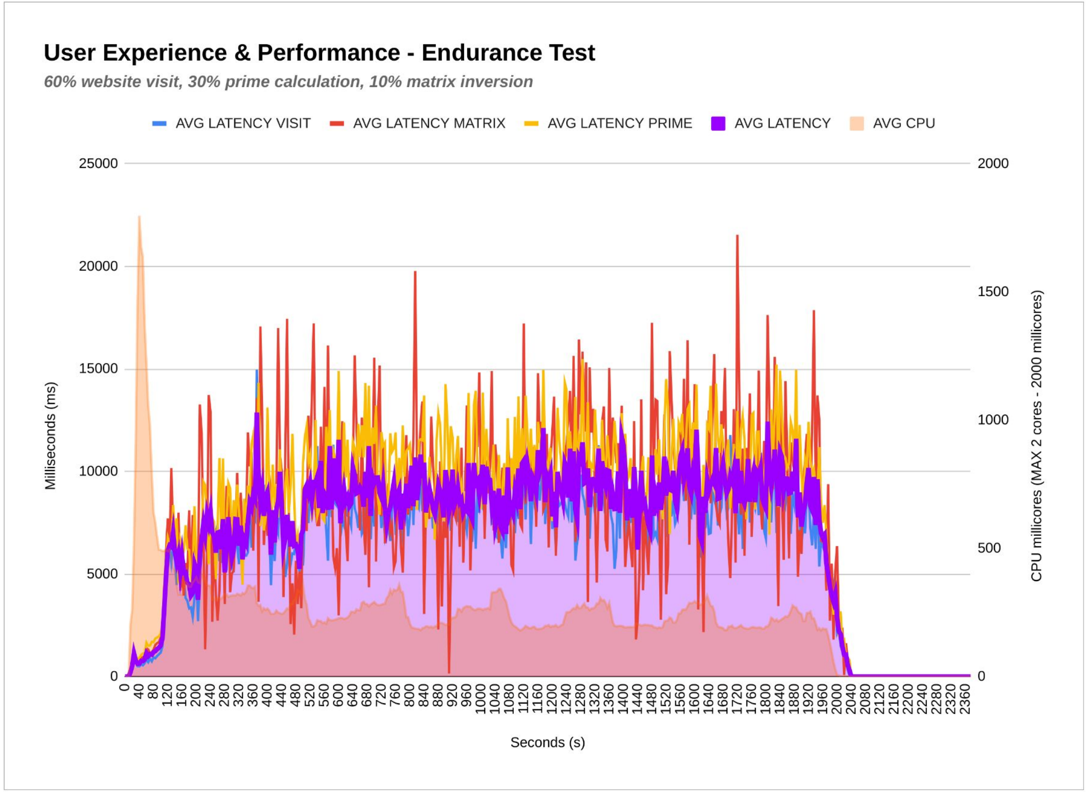
Experiment 1 - Load Test



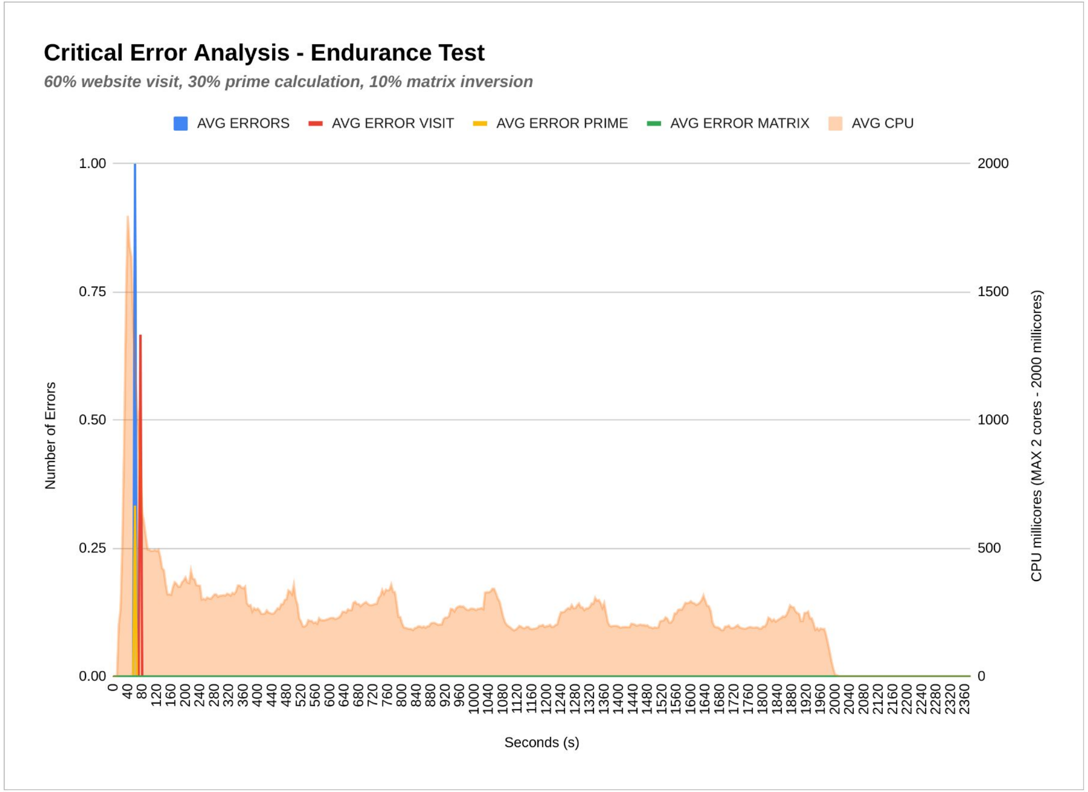
Experiment 1 - Endurance Test



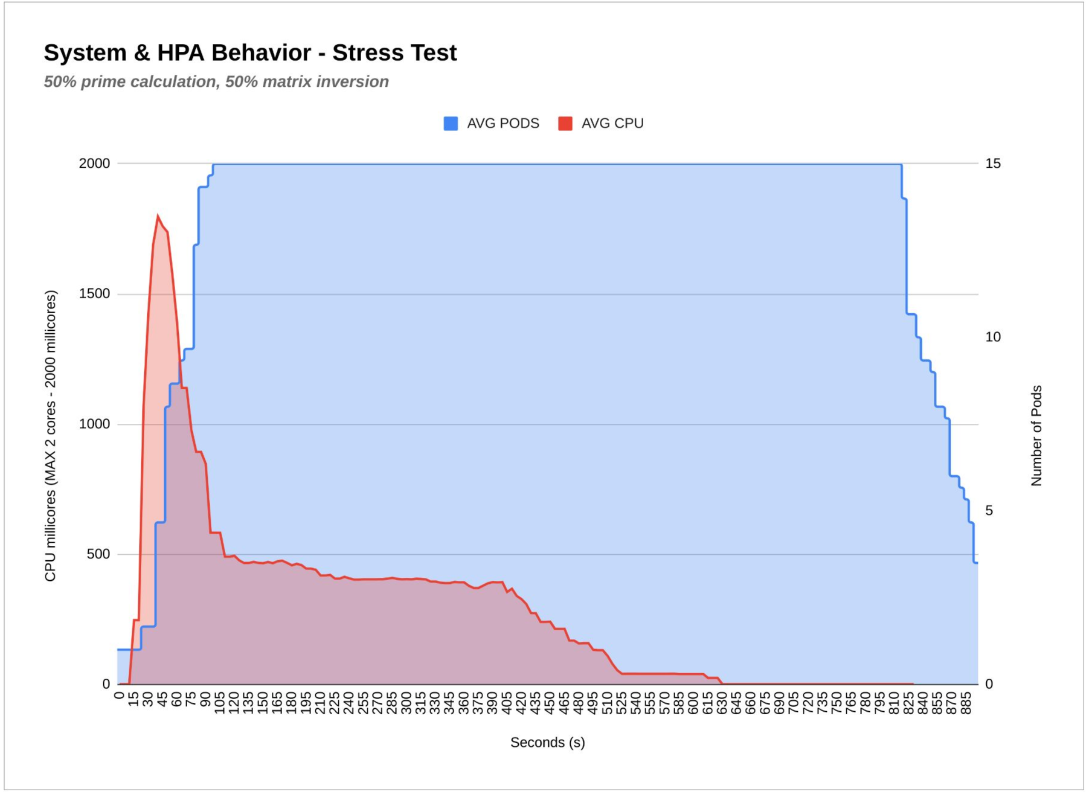
Experiment 1 - Endurance Test



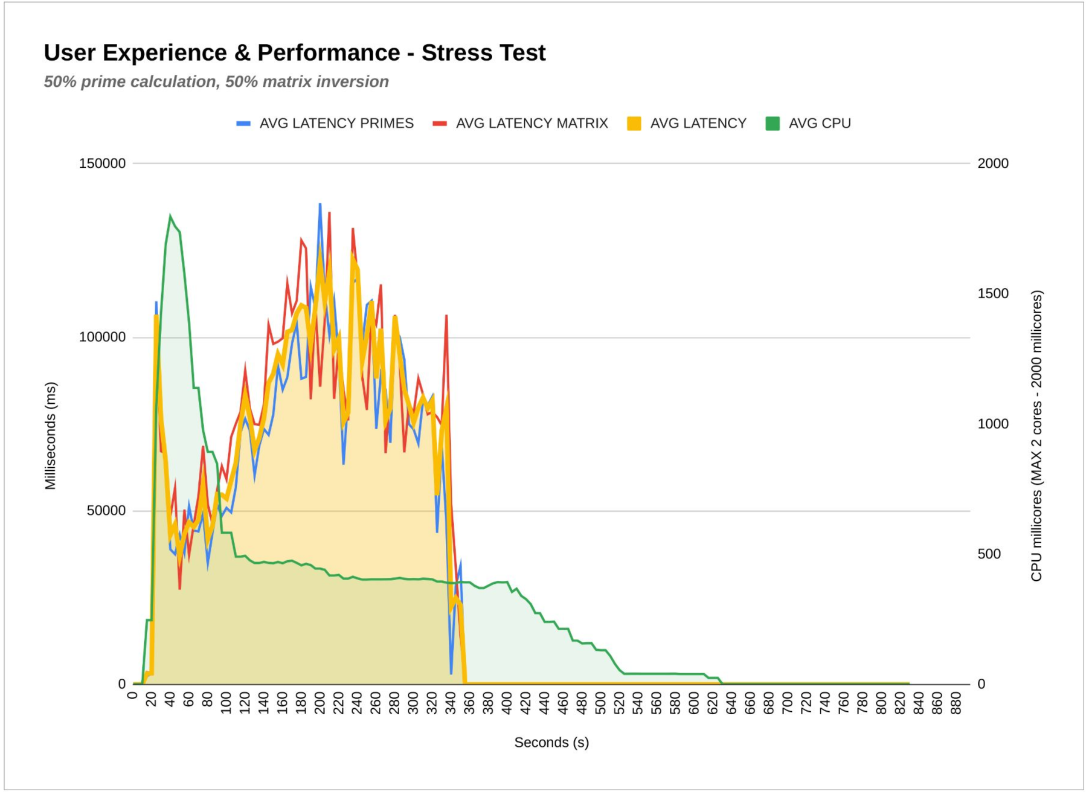
Experiment 1 - Endurance Test



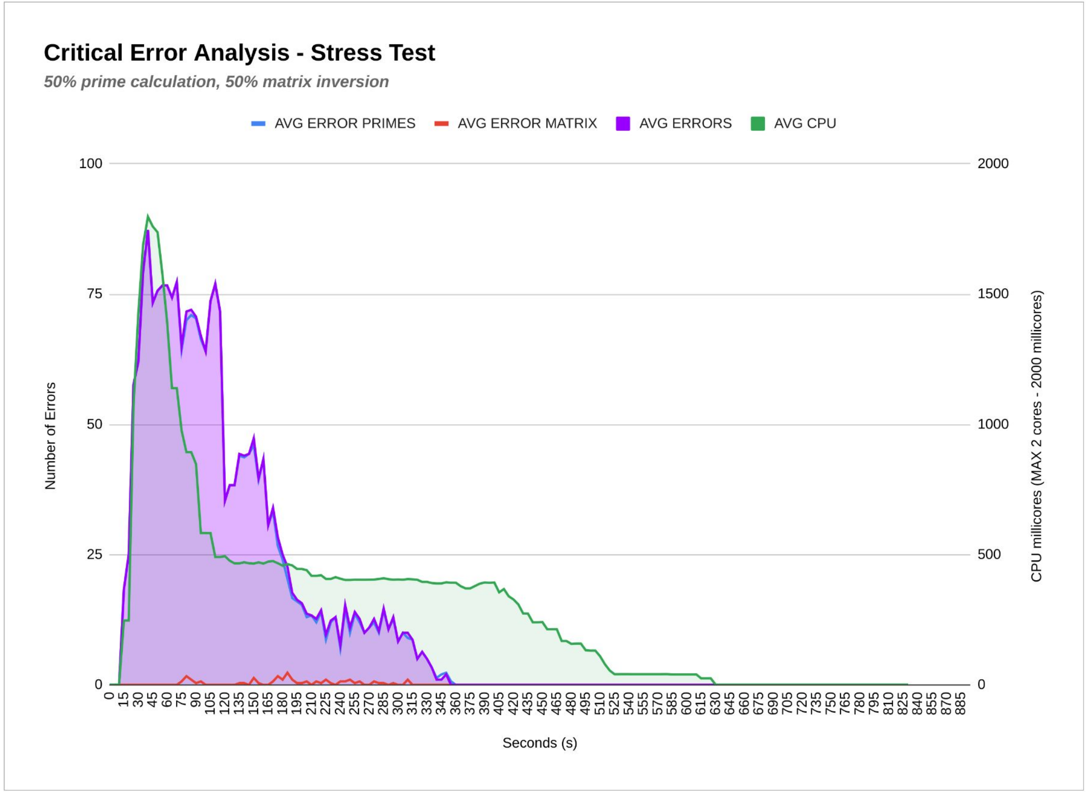
Experiment 1 - Stress Test



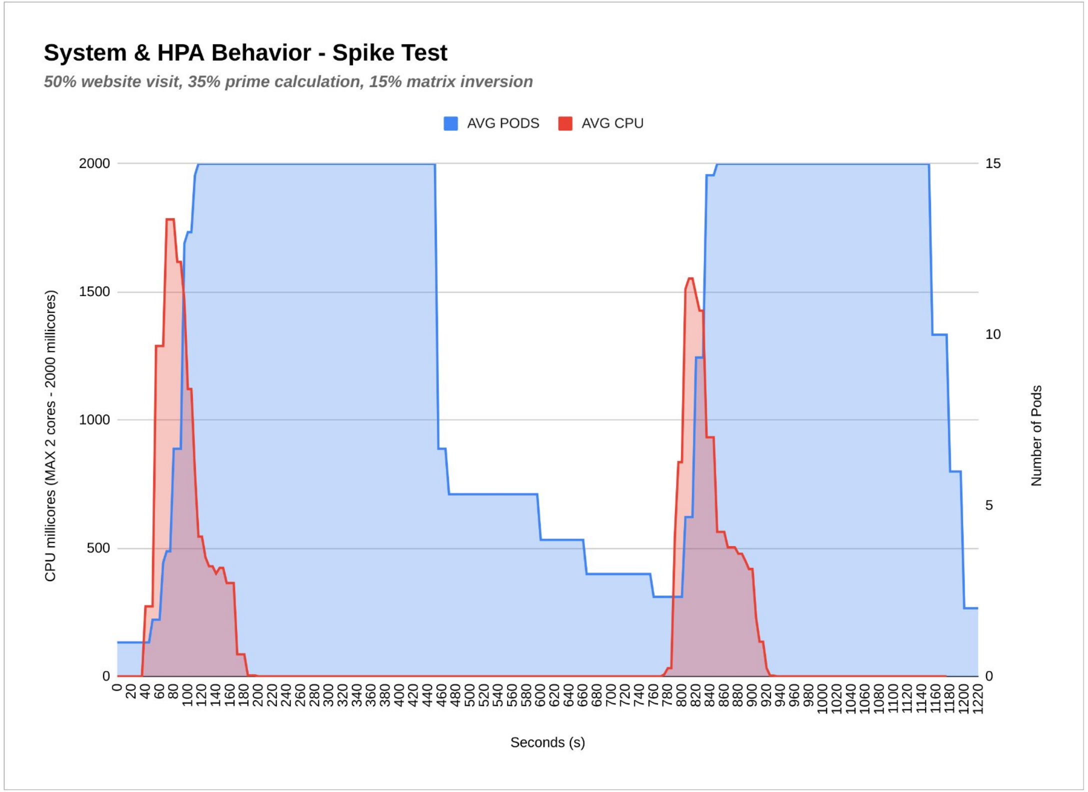
Experiment 1 - Stress Test



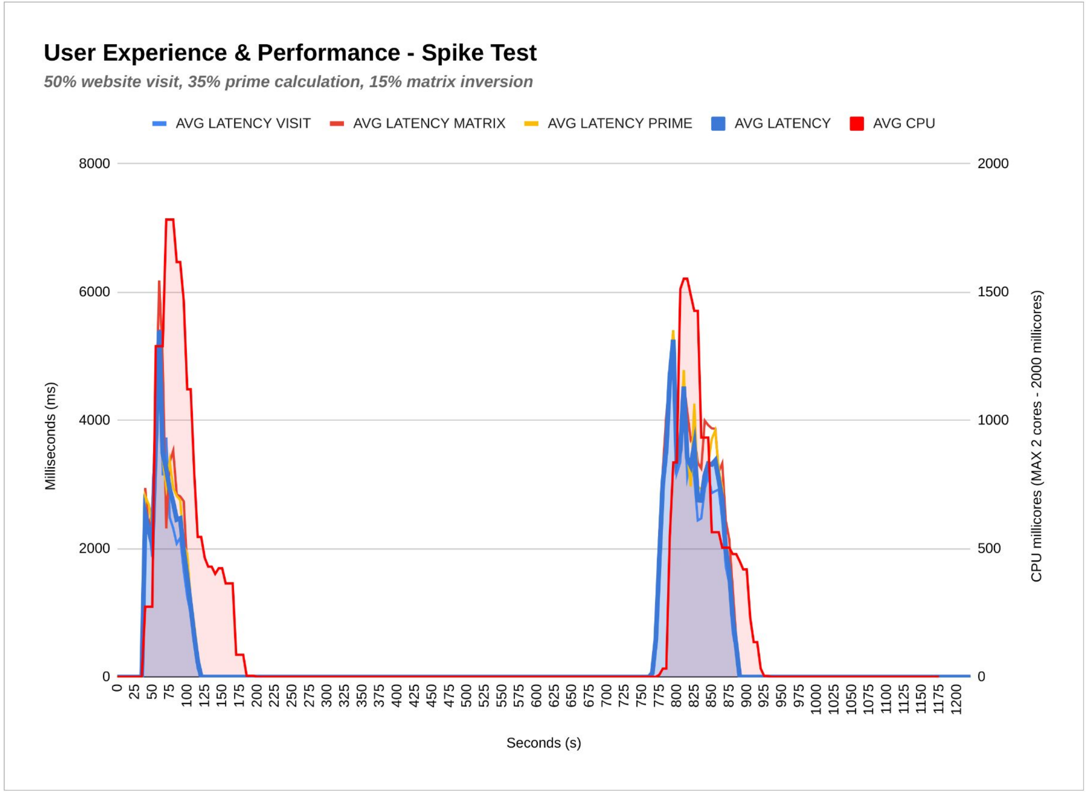
Experiment 1 - Stress Test



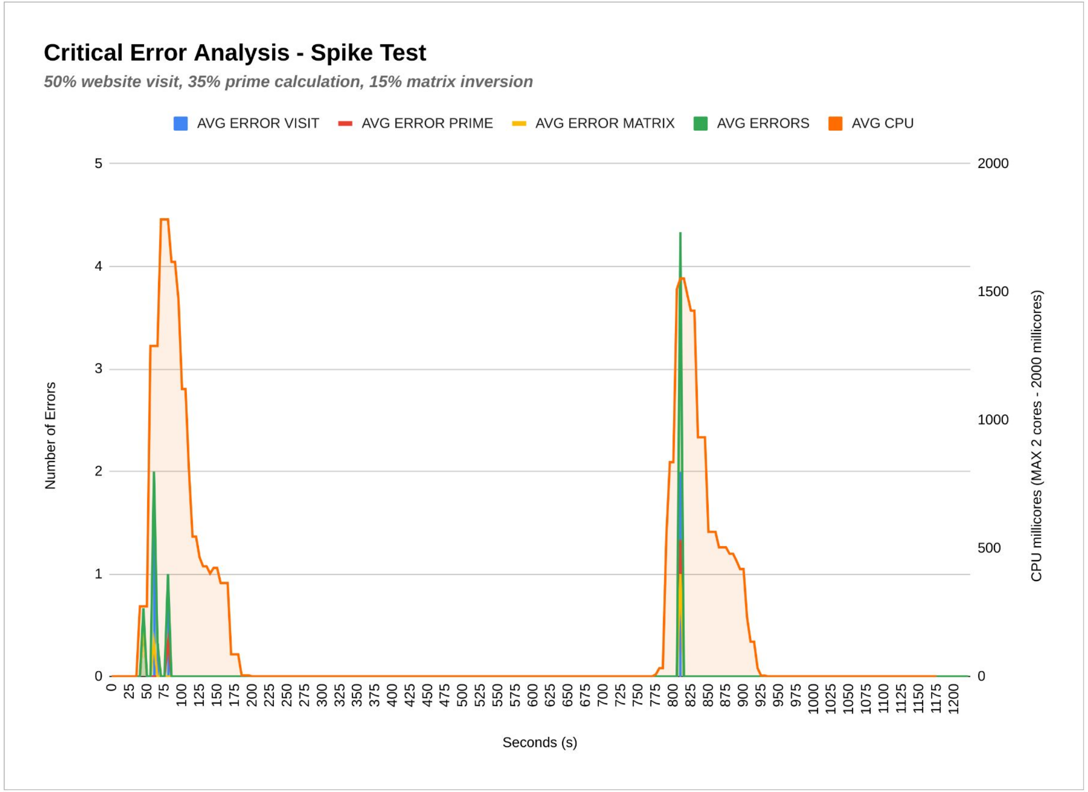
Experiment 1 - Spike Test



Experiment 1 - Spike Test

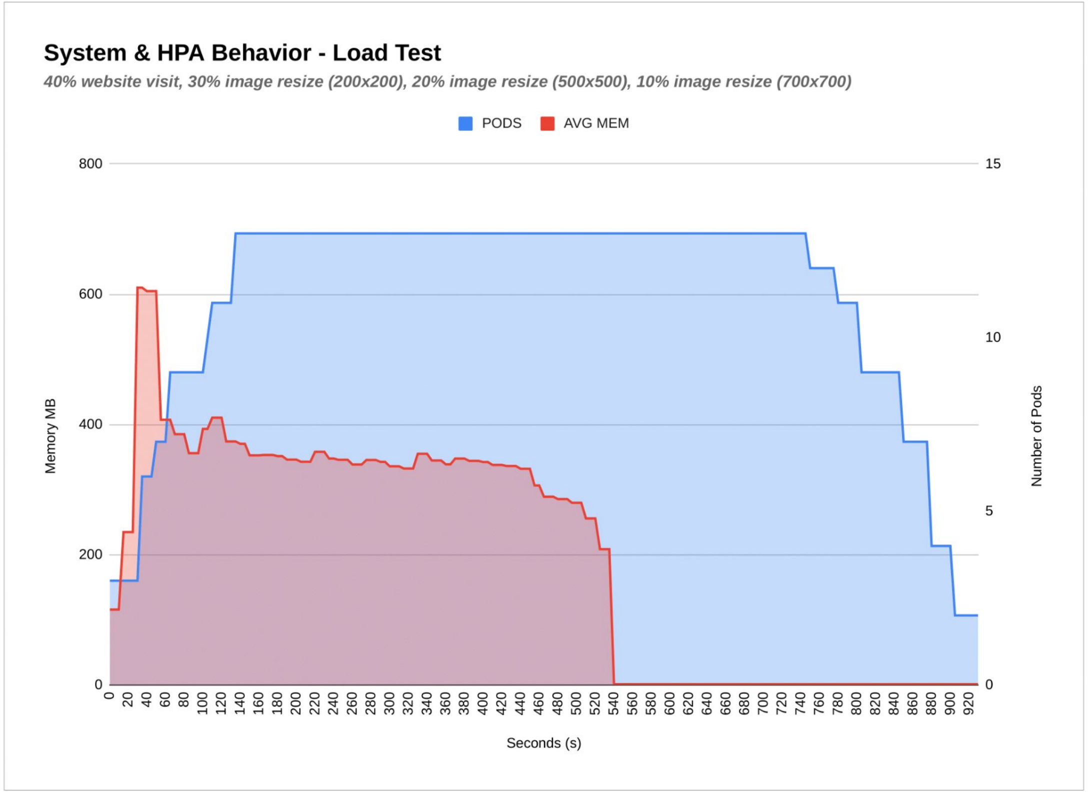


Experiment 1 - Spike Test

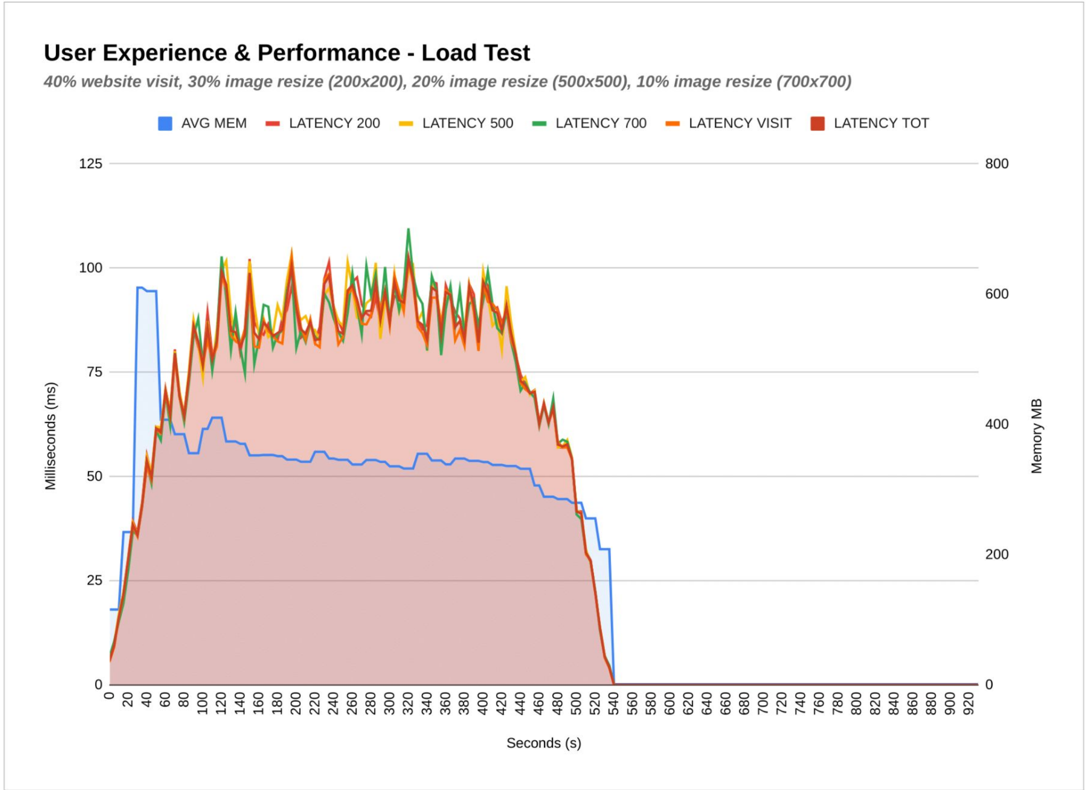


Experiment 2
Memory-Intensive

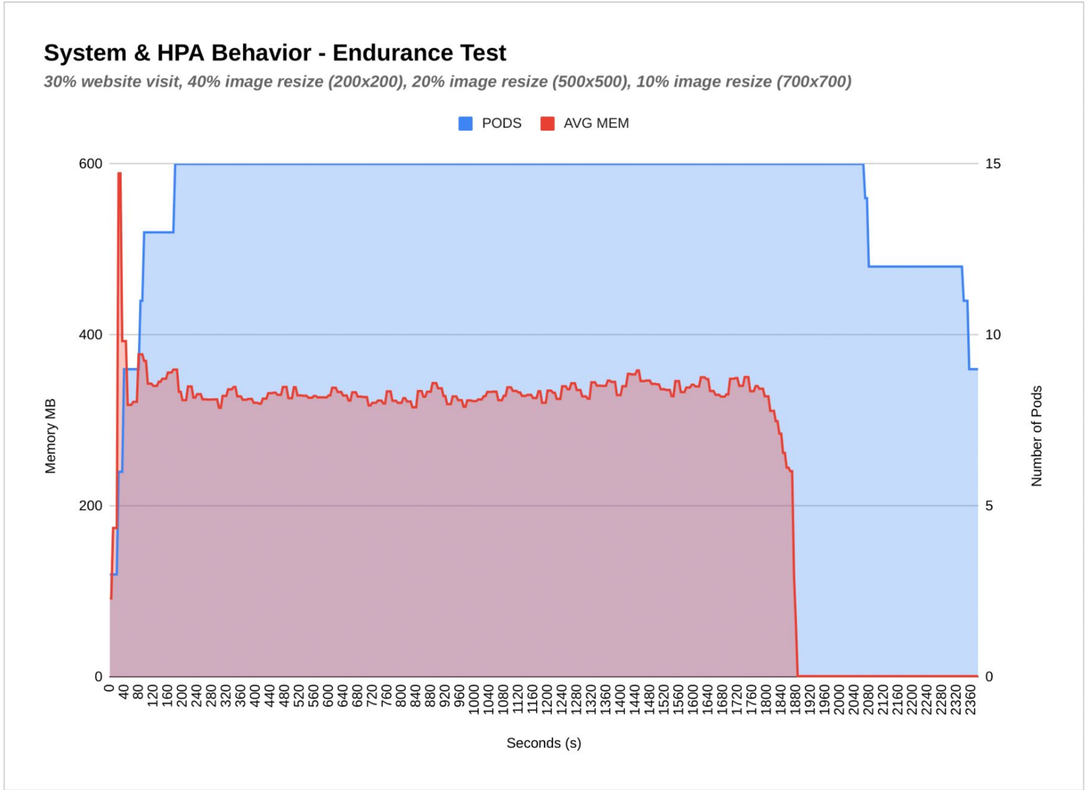
Experiment 2 - Load Test



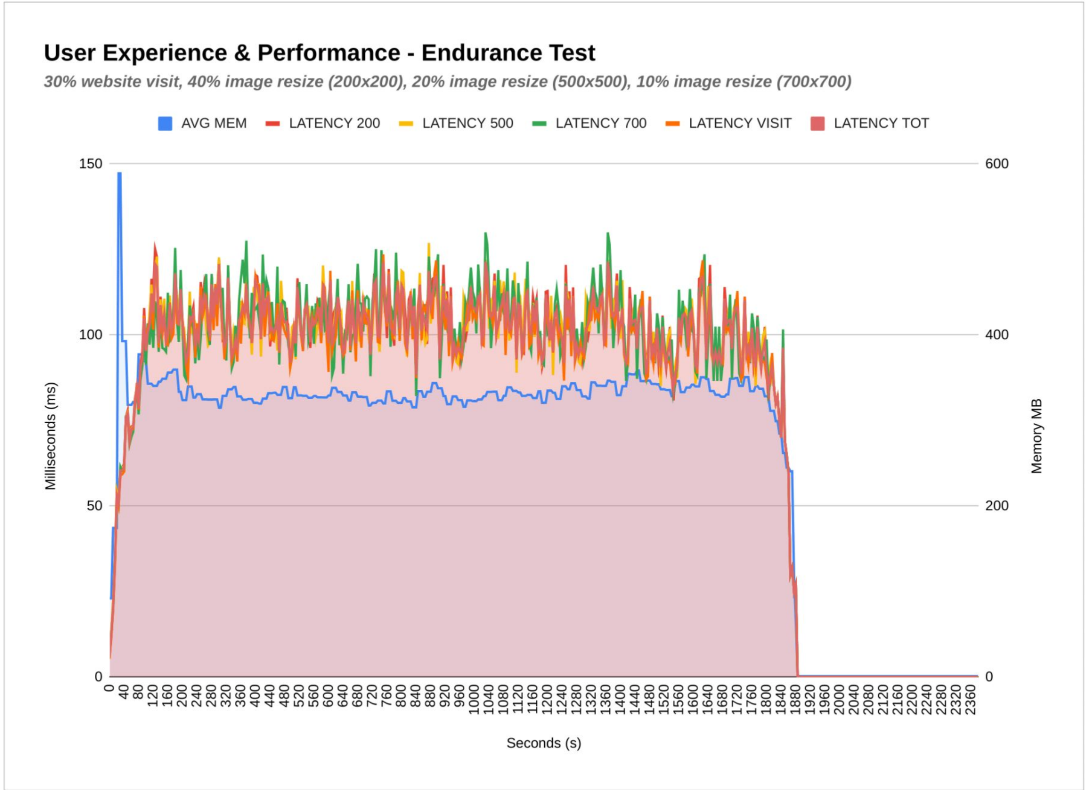
Experiment 2 - Load Test



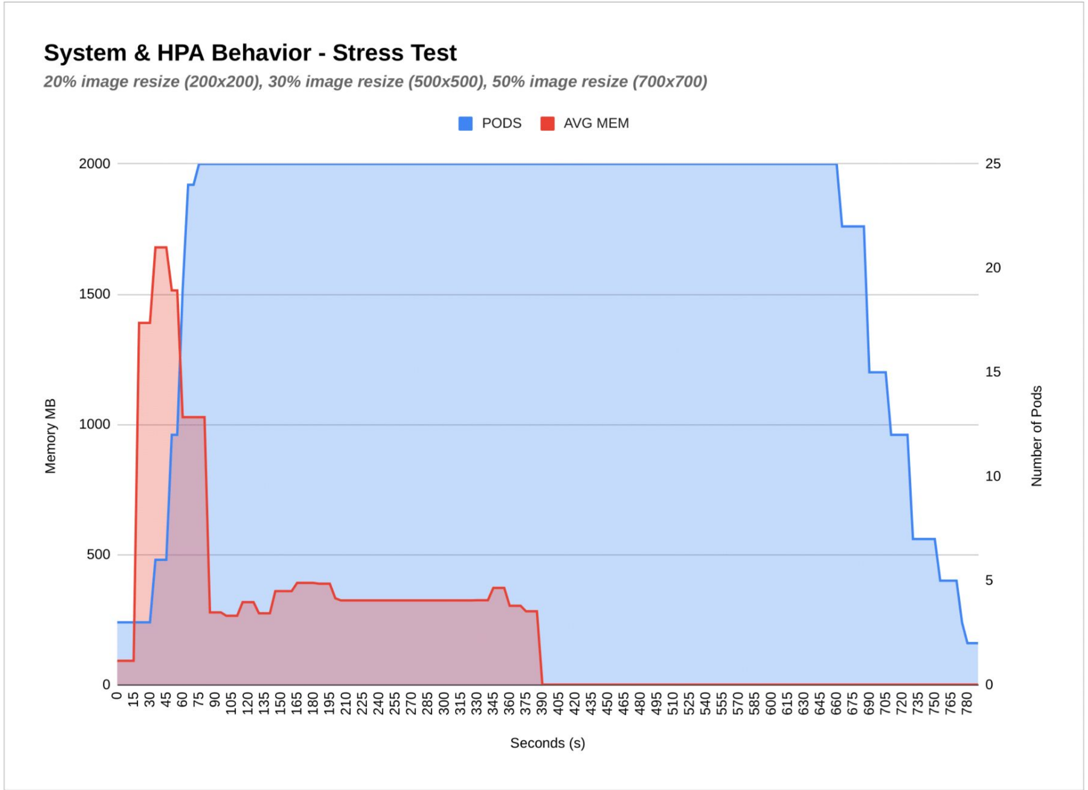
Experiment 2 - Endurance Test



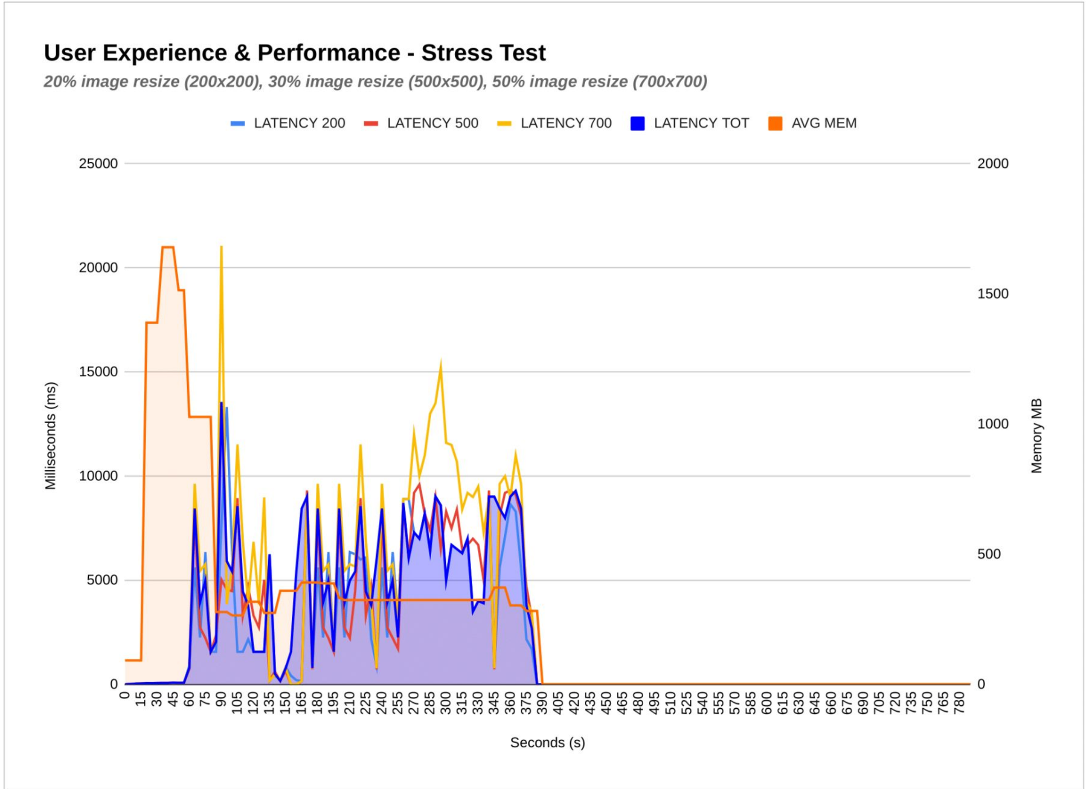
Experiment 2 - Endurance Test



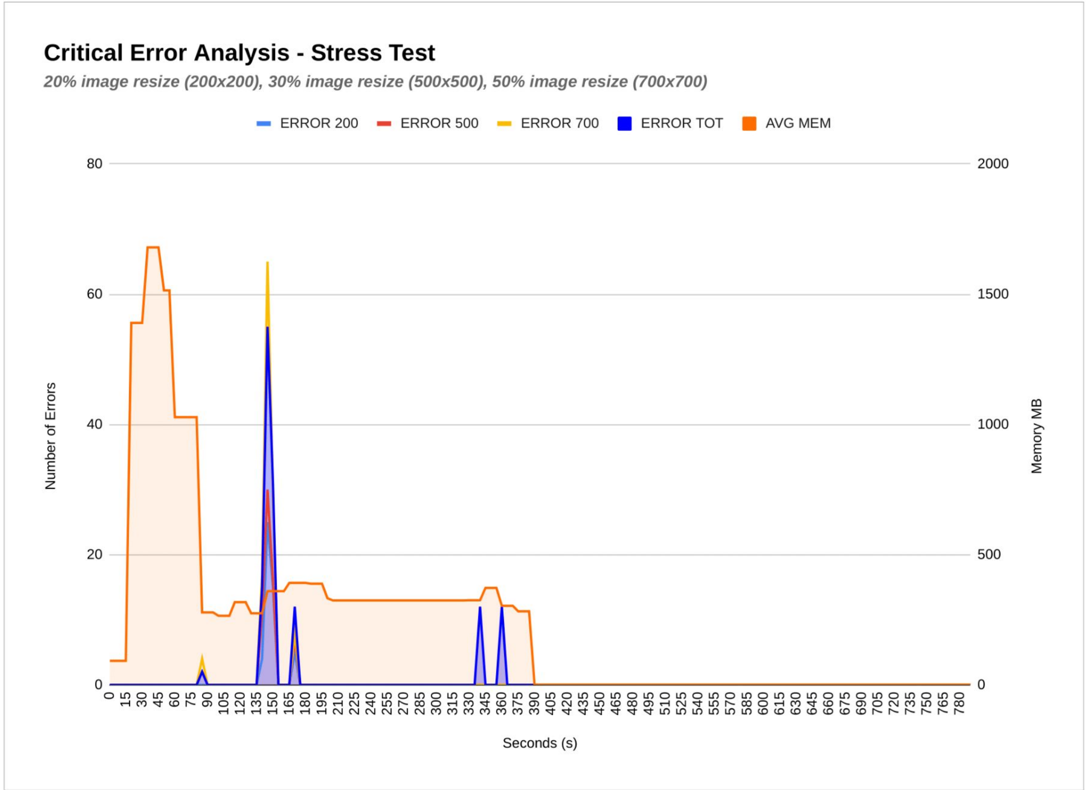
Experiment 2 - Stress Test



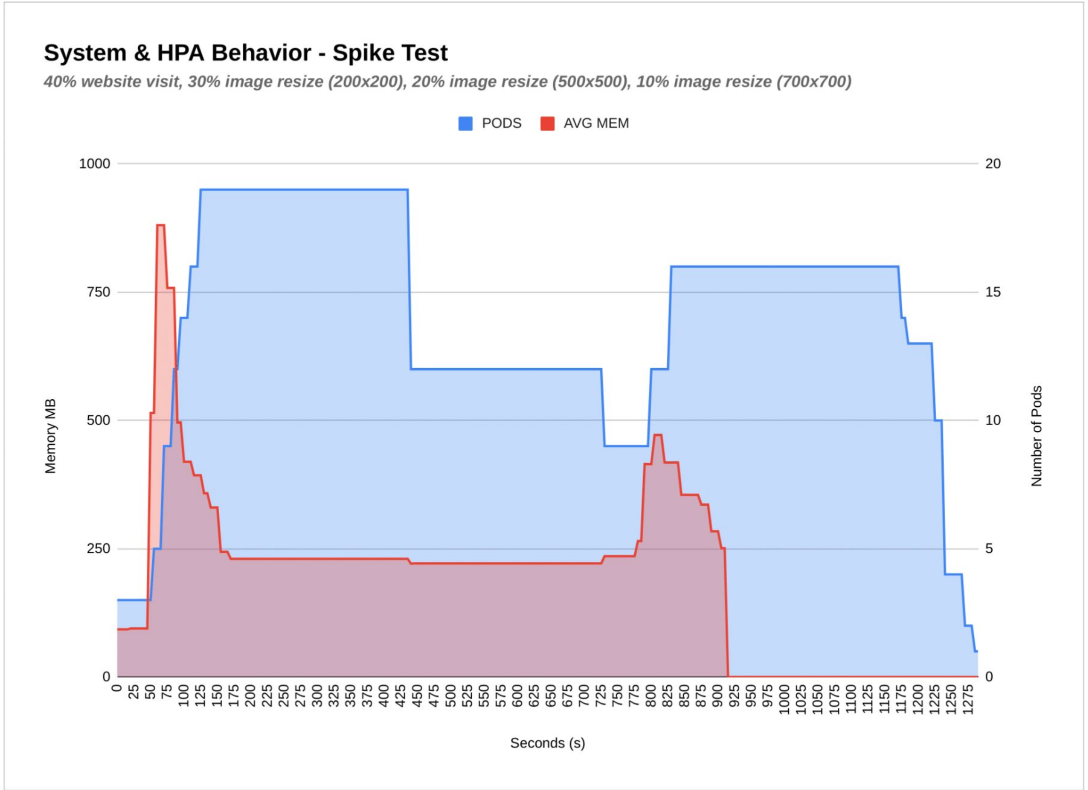
Experiment 2 - Stress Test



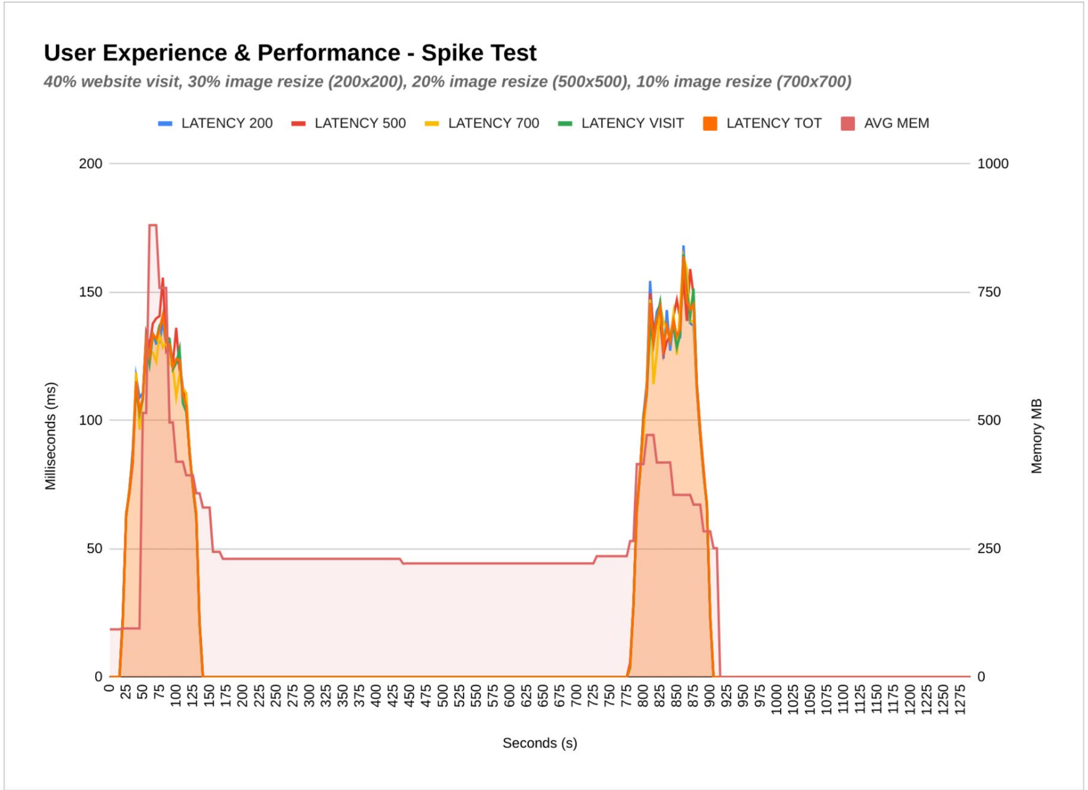
Experiment 2 - Stress Test



Experiment 2 - Spike Test



Experiment 2 - Spike Test



Results Analysis

Experiment 1 - CPU Intensive

Test Type	HPA Behavior	User Experience
Load Test	Scaled up pods correctly	High latency (>4s); system struggled
Endurance Test	Scaled aggressively to max capacity	**System Failure.** Latency >2 mins, high errors
Stress Test	Maintained a high pod count consistently for 30+ mins	Stable performance, no degradation over time
Spike Test	Reacted quickly to traffic bursts, scaled out & in	Elastic. Recovered quickly after initial latency

Experiment 2 - Memory Intensive

Test Type	HPA Behavior	User Experience
Load Test	Scaled perfectly based on memory	**Excellent!** Low latency (~100ms) and 0% errors
Endurance Test	Scaled aggressively to max capacity	**System Failure.** High errors due to Out-of-Memory
Stress Test	Maintained a high pod count consistently for 30+ mins.	Very reliable. 0% error rate over a long duration
Spike Test	Reacted instantly to memory spike usage, scaled out & in	Elastic and Resilient. Handled spikes with 0% errors

Conclusion

Conclusion
06

The **Horizontal Pod Autoscaler** is a powerful and reliable mechanism for automating scaling. It consistently reacted as expected to both CPU and Memory pressure across all test scenarios.

Our stress tests proved that simply adding more pods (**scaling out horizontally**) is ultimately **constrained** by the **total CPU and Memory capacity** of the underlying worker nodes.

While the system **scaled correctly**, it didn't always maintain good performance for the user. When the nodes became **saturated**, **latency increased** substantially and **errors occurred**, even with the maximum number of pods running.

Thanks