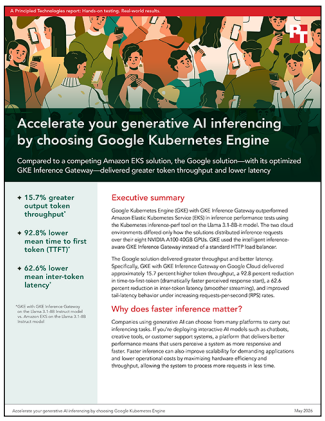

The science behind the report:

Accelerate your generative AI inferencing by choosing Google Kubernetes Engine

This document describes what we tested, how we tested, and what we found. To learn how these facts translate into real-world benefits, read the report Accelerate your generative AI inferencing by choosing Google Kubernetes Engine.

We concluded our hands-on testing on April 14, 2026. During testing, we determined the appropriate hardware and software configurations and applied updates as they became available. The results in this report reflect configurations that we finalized on March 12, 2026 or earlier. Unavoidably, these configurations may not represent the latest versions available when this report appears.

Our results

To learn more about how we have calculated the wins in this report, go to https://facts.pt/calculating-and-highlighting-wins. Unless we state otherwise, we have followed the rules and principles we outline in that document.

Table 1: Output token throughput and normalized output token latency (NTPOT) for testing on Google Kubernetes Engine. Source: PT.

Request rate	Output token throughput	Latency (milliseconds)	
Requests/second	Tokens/second Higher is better	Normalized time per output token (TPOT) Lower is better	
	Mean	Mean	P95
9.93	609.27	15.55	18.87
41.06	2,508.59	16.07	18.72
59.12	3,597.34	16.97	20.30
81.16	4,939.72	17.89	21.59
100.01	6,087.82	18.83	22.63
149.50	9,108.26	22.25	28.05
179.85	10,897.75	26.14	34.75

Table 2: Output token throughput and normalized output token latency (NTPOT) for testing on Amazon Elastic Kubernetes Service. Source: PT.

Request rate	Output token throughput	Latency (milliseconds)	
Requests/second	**Tokens/second** Higher is better	**Normalized time per output token (TPOT)** Lower is better	
	Mean	Mean	P95
9.91	610.68	16.98	21.85
40.17	2,455.55	18.69	24.46
61.05	3,714.97	21.29	28.43
79.81	4,855.87	24.81	33.91
102.34	6,166.06	31.36	42.68
152.18	8,613.18	116.49	169.60
154.25	8,228.22	169.45	216.36

As noted in the report, the inference-perf tool we used analyzes the throughput-latency tradeoff to compute the request rate that yields the highest output-token throughput before its latency starts to rapidly increase. The metrics in Table 2 are from that request rate.

Table 3: Metrics at throughput-latency trade-off point. Source: PT.

	Google Kubernetes Engine	Amazon Elastic Kubernetes Service
Output token throughput Mean tokens per second	7,169.21	6,042.05
Time to final token (TTFT) Mean latency (ms)	188.36	2624.73
Inter-token (ITL) Mean latency (ms)	30.29	81.03
Normalized time per output token (NTPOT) 95th percentile tail latency (ms)	79.29	240.57

System configuration information

Table 4: Detailed information on the systems we tested.

General information	Google solution	Amazon solution
Date testing ended	4/14/2026	4/14/2026
Cloud service provider (CSP)	Google Cloud with GKE Inference Gateway	Amazon EKS
Region	asia-northeast3	ap-northeast-2
Kubernetes version	v1.35.0-gke.1795000	1.35.1
VLLM image	vllm-openai:v0.11.0	vllm-openai:v0.11.0
GKE Inference Gateway version	v1.2.1	N/A
Inference Workload information		
Workload client and version	inference-perf v0.2.0	
Workload 1	Chat with SharedGPT prompts	
Workload 2	Random prompts with correlated prefixes	
Iterations and result choice	3 test runs, median reported	
Cloud VMs for accelerator nodes		
Number of VMs	1	1
VM or instance size	a2-highgpu-8g	p4d.24xlarge
VM image	cos-125-19216-104-32	ami-08123720165f78d95
vCPU	96	96
Memory (GB)	680	1,152
Underlying processor model	Intel® Xeon® Platinum 8273CL	Intel Xeon Platinum 8275L
Max. network bandwidth (Gbps)	100	400
Additional storage (GB)	1,280	1,300
Accelerators	NVIDIA® A100 (40GB)	NVIDIA A100 (40GB)
Number of accelerators	8	8
High bandwidth memory (GB)	40	40
NVIDIA driver version	580.105.0	580.95.05
Cloud VMs for inference client nodes		
Number of VMs	1	1
VM machine / instance	n2-standard-16	m6i.4xlarge
VM image	cos-125-19216-104-32	ami-0ebee15b3831418f2
vCPUs	16	16
Memory (GB)	64	64
Underlying processor model	Intel Ice Lake or Cascade Lake	Intel Xeon 8375C
Max. network bandwidth (Gbps)	32	12.5
Additional storage	N/A	N/A
Cloud VMs for worker nodes		
Number of VMs	3	3
VM machine / instance	n2-standard-4	m6i.xlarge
VM image	cos-125-19216-104-32	ami-0ebee15b3831418f2
vCPU	2	4
Memory (GB)	8	16
Underlying processor model	Intel Ice Lake or Cascade Lake	Intel Xeon 8375C
Max. network bandwidth (Gbps)	10	12.5
Additional storage	N/A	N/A

How we tested

Preparing the testing environment on Google Cloud Platform

1. Prepare the CLI environment and configure the project's global settings:

```
curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | \
  sudo gpg --dearmor -o /usr/share/keyrings/cloud.google.gpg
echo "deb [signed-by=/usr/share/keyrings/cloud.google.gpg] https://packages.cloud.google.com/apt
cloud-sdk main" | \
  sudo tee -a /etc/apt/sources.list.d/google-cloud-sdk.list
sudo apt update && sudo apt install google-cloud-cli
gcloud init
wget https://get.helm.sh/helm-v3.18.6-linux-amd64.tar.gz
tar -zxvf ~/Downloads/helm-v3.18.6-linux-amd64.tar.gz
sudo install linux-amd64/helm  /usr/local/bin/
wget https://dl.k8s.io/release/v1.33.1/bin/linux/amd64/kubectl
sudo install kubectl /usr/local/bin
```

2. Use shell variables to define the environment:

```
export IGW_VERSION=v1.4.0
export HF_TOKEN="hf_XXXXXX"
export USER_EMAIL="XXXX@XXXX.com"
export PROJECT_ID="XXXX"
export PROJECT_NUMBER="XXXX"
export REGION="asia-northeast3"
export SUB_REGION="b"
export CONTROL_PLANE_LOCATION="$REGION"
export PROXY_SUBNET_RANGE="10.120.0.0/23"
export VPC_NETWORK_NAME="default"
export PROXY_SUBNET_NAME="proxy-only-subnet-${REGION}"
export CLUSTER_NAME="llm-inference-cluster"
export WORKER_MACH="n2-standard-8"
export CLIENT_MACH="n2-standard-16"
export ACCELERATOR_POOL_NAME="a100-pool"
export ACCELERATOR_ZONES="${CONTROL_PLANE_LOCATION}-${SUB_REGION}"
export GPU_COUNT=8
export GPU_TYPE="nvidia-tesla-a100"
export ACCEL_MACH="a2-highgpu-8g"
```

3. Prepare the GCP CLI global configuration:

```
gcloud config set compute/region $REGION
gcloud config list compute/region
gcloud config set core/account [insert email address]
gcloud config set project "$PROJECT_ID"
gcloud config set billing/quota_project "$PROJECT_ID"
gcloud config set compute/region $REGION
gcloud config configurations create YYY-project
```

4. Create the Kubernetes cluster:

```
gcloud compute networks subnets create $PROXY_SUBNET_NAME \
  --purpose=REGIONAL_MANAGED_PROXY \
  --role=ACTIVE \
  --region=$REGION \
  --network=$VPC_NETWORK_NAME \
  --range=$PROXY_SUBNET_RANGE \
  --project=$PROJECT_ID
gcloud container clusters create "$CLUSTER_NAME" \
  --project="$PROJECT_ID" \
  --location="$CONTROL_PLANE_LOCATION" \
  --workload-pool="$PROJECT_ID".svc.id.goog \
  --release-channel=rapid \
  --machine-type="$WORKER_MACH" \
  --num-nodes=1 \
  --enable-managed-prometheus \
  --monitoring=SYSTEM,DEPLOYMENT,HPA,POD,DCGM \
  --auto-monitoring-scope=ALL \
  --gateway-api=standard
```

5. Create a Google Cloud Storage bucket to store testing results and allow access from Kubernetes:

```
gcloud storage buckets create gs://cbfgf-bucket --location=US --uniform-bucket-level-access
kubectl create serviceaccount inf-perf
for role in storage.expressModeUserAccess storage.objectCreator ; do
  gcloud storage buckets add-iam-policy-binding gs:// cbfgf-bucket -bucket/ \
    --role="roles/$role" \
    --member="principal://iam.googleapis.com/projects/${PROJECT_NUMBER}/locations/global/
workloadIdentityPools/${PROJECT_ID}.svc.id.goog/subject/ns/default/sa/inf-perf" \
    --condition=None
done
```

6. Add the A100 (40GB) accelerator VM to the cluster:

```
gcloud container node-pools create "$ACCELERATOR_POOL_NAME" \
  --cluster=${CLUSTER_NAME} \
  --project=${PROJECT_ID} \
  --location=${CONTROL_PLANE_LOCATION} \
  --node-locations=${ACCELERATOR_ZONES} \
  --machine-type=${ACCEL_MACH} \
  --disk-size=500 \
  --enable-autoscaling \
  --min-nodes=0 \
  --max-nodes=1 \
  --spot
```

7. Deploy eight vLLM instances:

```
# Huggingface Token for inference-perf and vllm
kubectl create secret generic hf-secret --from-literal=hf_api_token="${HF_TOKEN}" \
 --dry-run=client -o yaml | kubectl apply -f -
kubectl create secret generic hf-token --from-literal=token="${HF_TOKEN}"
 --dry-run=client -o yaml | kubectl apply -f -
# The file "vllm-llama31-8B.yaml" is found in the appendix
kubectl apply -f vllm-llama31-8B.yaml
```

8. Deploy IGW application:

```
helm template llama-pool --dependency-update \
  --set inferencePool.modelServers.matchLabels.app=llama-pool \
  --set provider.name=gke --version "${IGW_VERSION}" \
  --set experimentalHttpRoute.enabled=true \
  oci://registry.k8s.io/gateway-api-inference-extension/charts/inferencepool \
> igw-template.yaml
# patch the default template so that the the scorers have equal weights
patch << EOF
--- igw-template.yaml-orig 2026-02-11 14:31:22.527363716 -0400
+++ igw-template.yaml      2026-02-11 14:33:32.996625151 -0400
@@ -31,7 +31,7 @@
       - pluginRef: kv-cache-utilization-scorer
         weight: 2
       - pluginRef: prefix-cache-scorer
-        weight: 3
+        weight: 2
 ---
 # Source: inferencepool/templates/rbac.yaml
 apiVersion: rbac.authorization.k8s.io/v1
EOF
# deploy IGW
kubectl apply -f igw-template.yaml
echo "Waiting for Gateway IP address..."
kubectl wait --for=condition=Programmed=True gateway/inference-internal-gateway \
  --timeout=10m
export IP=$(kubectl get gateway/inference-internal-gateway \
  -o jsonpath='{.status.addresses[0].value}' 2>/dev/null)
echo "Gateway IP Address is $IP"
```

9. Deploy the VM for the inference-perf client:

```
gcloud container node-pools create client-pool \
    --cluster="$CLUSTER_NAME" \
    --project="$PROJECT_ID" \
    --workload-metadata=GKE_METADATA \
    --node-locations="$ACCELERATOR_ZONES" \
    --location="$CONTROL_PLANE_LOCATION" \
    --machine-type="$CLIENT_MACH" \
    --num-nodes=1
```

Preparing the testing environment on Amazon Web Services

1. Prepare the CLI environment and configure the project's global settings:

```
ARCH=amd64
PLATFORM=$(uname -s)_$ARCH
wget "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_$PLATFORM.tar.gz"
curl -sL https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_checksums.txt\
  | grep $PLATFORM | sha256sum --check
tar -xf eksctl_$PLATFORM.tar.gz -C /tmp && rm eksctl_$PLATFORM.tar.gz
sudo install -m 0755 /tmp/eksctl /usr/local/bin && rm /tmp/eksctl
aws sts get-caller-identity --profile XXXXX
aws sso login
wget https://get.helm.sh/helm-v3.18.6-linux-amd64.tar.gz
tar -zxvf ~/Downloads/helm-v3.18.6-linux-amd64.tar.gz
sudo install linux-amd64/helm /usr/local/bin/
wget https://dl.k8s.io/release/v1.33.1/bin/linux/amd64/kubectl
sudo install kubectl /usr/local/bin
```

2. Use environment variables to define the environment:

```
export K8S_VERSION=1.34
export AWS_REGION="us-east-1"
export CLUSTER=vllm-cluster
export NVIDIA_AMI="$(aws ssm get-parameter --name /aws/service/bottlerocket/aws-k8s-${K8S_VERSION}-
nvidia/x86_64/latest/image_id --query Parameter.Value --output text)"
export STANDARD_AMI="$(aws ssm get-parameter --name /aws/service/bottlerocket/aws-k8s-${K8S_
VERSION}/x86_64/latest/image_id --query Parameter.Value --output text)"
export ACCOUNT_ID="$(aws sts get-caller-identity --query Account --output text)"
export CLIENT_MACH= m6i.4xlarge
```

3. Create the Kubernetes cluster:

```
set -u
# see the file cluster.yaml, below
envsubst <<EOF | eksctl create cluster -f -
# File: cluster.yaml
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: $CLUSTER
  region: $AWS_REGION
  version: "$K8S_VERSION"
autoModeConfig:
  enabled: false
cloudWatch:
  clusterLogging:
    enableTypes:
      - api
      - audit
      - authenticator
      - controllerManager
      - scheduler
addons:
  - name: vpc-cni
    version: latest
  - name: coredns
    version: latest
  - name: kube-proxy
    version: latest
managedNodeGroups:
  - name: vllm-cluster-nodes
    instanceType: ${CLIENT_MACH}
    ami: $STANDARD_AMI
    amiFamily: Bottlerocket
    desiredCapacity: 3
    labels:
      role: cpu-worker
    tags:
      nodegroup-role: cpu-worker
    iam:
      withAddonPolicies:
        cloudWatch: true
EOF
```

4. Prepare the AWS global configuration:

```
eksctl utils associate-iam-oidc-provider --region="$AWS_REGION" --cluster=vllm-cluster --approve
```

5. Create an Amazon Simple Storage Service (S3) bucket to store testing results and allow access from Kubernetes:

```
aws s3api create-bucket --bucket cbfgf-bucket -bucket --region "$AWS_REGION" \
  --create-bucket-configuration LocationConstraint="$AWS_REGION"
# create an OICD provider
eksctl utils associate-iam-oidc-provider --cluster "$CLUSTER" --approve
# check
aws iam list-open-id-connect-providers
# create an S3 access policy
aws iam create-policy --policy-name s3-policy --policy-document file://s3-access-policy.json
# add a SA
kubectl apply -f s3-access-sa.yaml
# create a policy document
oidc_provider=$(aws eks describe-cluster --name "$CLUSTER" --region "$AWS_REGION" --query "cluster.
identity.oidc.issuer" --output text | sed -e "s/^https:\/\///")
service_account=s3-access
namespace=default
##### if necesary, delete previous roles and policies
rm -f s3-trust-policy.json
aws iam detach-role-policy --role-name s3-role \
  --policy-arn="arn:aws:iam::$ACCOUNT_ID:policy/s3-policy"
aws iam delete-role --role-name s3-role
## create new policy, role, and attach it.
cat >s3-trust-policy.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::$ACCOUNT_ID:oidc-provider/$oidc_provider"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "$oidc_provider:aud": "sts.amazonaws.com",
          "$oidc_provider:sub": "system:serviceaccount:$namespace:$service_account"
        }
      }
    }
  ]
}
EOF
# create a role with with trust policy
aws iam create-role --role-name s3-role --assume-role-policy-document \
  file://s3-trust-policy.json --description "access s3 from EKS"
# attack the IAM policy to this role
aws iam attach-role-policy --role-name s3-role \
  --policy-arn="arn:aws:iam::$ACCOUNT_ID:policy/s3-policy"
# annotate
kubectl annotate serviceaccount -n "$namespace" "$service_account" \
  eks.amazonaws.com/role-arn="arn:aws:iam::$ACCOUNT_ID:role/s3-role"
```

6. Add the A100 (40GB) accelerator VM to the cluster:

```
envsubst <<EOF | eksctl create nodegroup -f -
# File: accel.yaml
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: $CLUSTER
  region: $AWS_REGION
  version: "$K8S_VERSION"
autoModeConfig:
  enabled: false
addons:
  - name: vpc-cni
    version: latest
  - name: coredns
    version: latest
```

```
  - name: kube-proxy
    version: latest
managedNodeGroups:
  - name: vllm-p4d-nodes
    instanceType: p4d.24xlarge
    desiredCapacity: 1
    volumeSize: 1300
    privateNetworking: true
    ami: $NVIDIA_AMI
    amiFamily: Bottlerocket
    labels:
      role: large-model-worker
      nvidia.com/gpu: "true"
      k8s.amazonaws.com/accelerator: nvidia-gpu
    tags:
      nodegroup-role: large-model-worker
EOF
```

7. Deploy eight vLLM instances:

```
# Huggingface Token for inference-perf and vllm
kubectl create secret generic hf-secret --from-literal=hf_api_token="${HF_TOKEN}" \
 --dry-run=client -o yaml | kubectl apply -f -
kubectl create secret generic hf-token --from-literal=token="${HF_TOKEN}"
 --dry-run=client -o yaml | kubectl apply -f -
# The file "vllm-llama31-8B.yaml" is found in the appendix
kubectl apply -f vllm-llama31-8B.yaml
```

8. Install and configure the front-end load balancer:

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.14.0/docs/
install/iam_policy.json
aws iam create-policy \
  --policy-name AWSLoadBalancerControllerIAMPolicy \
  --policy-document file://iam_policy.json
eksctl create iamserviceaccount \
  --cluster="$CLUSTER" \
  --namespace=kube-system \
  --name=aws-load-balancer-controller \
  --attach-policy-arn="arn:aws:iam::$ACCOUNT_ID:policy/AWSLoadBalancerControllerIAMPolicy" \
  --override-existing-serviceaccounts \
  --region "$AWS_REGION" \
  --approve
helm repo add eks https://aws.github.io/eks-charts
helm repo update eks
helm install aws-load-balancer-controller eks/aws-load-balancer-controller \
  -n kube-system \
  --set clusterName="$CLUSTER" \
  --set serviceAccount.create=false \
  --set serviceAccount.name=aws-load-balancer-controller \
  --version 1.13.0
```

9. Deploy VM for the inference-perf client:

```
envsubst <<EOF | eksctl create nodegroup -f -
# File: client.yaml
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: $CLUSTER
  region: $AWS_REGION
  version: "$K8S_VERSION"

managedNodeGroups:
  - name: vllm-cluster-client
    instanceType: m6i.4xlarge
    ami: $STANDARD_AMI
    amiFamily: Bottlerocket
    desiredCapacity: 1
    labels:
      role: client
    tags:
      nodegroup-role: client
    iam:
      withAddonPolicies:
        cloudWatch: true
```

Conducting the tests

1. Load the configuration for the use-case. You must set three shell variables:

```
# URL => the URL for the inference gateway or the lod balancer
# BUCKET_TYPE => set of the bucket type; viz.,
# For EKS, set it to     simple_storage_service
# For GKE, set it to     google_cloud_storage
# HF_TOKEN => the HUggubgFace access token, identical to that yused above
#The configureation file "config.yaml" is in the appendix
# Create the realized configuration file from the template, expanding the three shell variables.

kubectl create configmap inference-perf-config \
  --from-file=config.yml
```

2. Run inference-perf as a Kubernetes job:

```
kubectl apply -f ip-manifest.yaml
```

3. Once the job finishes, copy test data to the local directory. Download the job log from the cluster:

```
kubectl logs job/inference-perf > client_logs.txt
```

4. Download the test results from the CSP bucket:
   a. For EKS, use this command:

```
aws s3  cp --recursive s3://cbfgf-bucket/$P
```

   b. For GKS, use this command:

```
gcloud storage cp gs://cbfgf-bucket/\* .
```

5. Extract the run data from the JSON files:

```
cat stage_* | \
  jq -r "[.load_summary.achieved_rate, \
  .successes.throughput.output_tokens_per_sec, \
  .successes.latency.normalized_time_per_output_token.mean, \
  .successes.latency.normalized_time_per_output_token.p95, \
  .successes.latency.normalized_time_per_output_token.p99]" \
  | jq -r '@csv'
jq -r "[.successes.latency.request_latency.p95, \
  .successes.latency.normalized_time_per_output_token.p95,.\
  successes.latency.time_per_output_token.p95, \
  .successes.latency.time_to_first_token.p95, \
  .successes.latency.inter_token_latency.p95]" summary_lifecycle_metrics.json \
  | jq -r @csv
```

YAML file used in testing

```
# File: vllm-llama31-8B.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: llama-pool
  name: llama-modelserver
  namespace: default
spec:
  replicas: 8
  selector:
    matchLabels:
      app: llama-pool
  template:
    metadata:
      labels:
        app: llama-pool
    spec:
      containers:
      - args:
        - --model
        - meta-llama/Llama-3.1-8B-Instruct
        - --tensor-parallel-size
        - "1"
        - --port
        - "8000"
        - --enable-prompt-tokens-details
        - --max-num-seqs
        - "128"
        - --max-model-len
        - "2048"
        command:
        - python3
        - -m
        - vllm.entrypoints.openai.api_server
        env:
        - name: HF_HOME
          value: /data
        - name: HUGGING_FACE_HUB_TOKEN
          valueFrom:
            secretKeyRef:
              key: token
              name: hf-token
        - name: LD_LIBRARY_PATH
          value: /usr/local/nvidia/lib64
        image: vllm/vllm-openai:v0.15.0
        imagePullPolicy: Always
        livenessProbe:
          failureThreshold: 5
          httpGet:
            path: /health
```

```
            port: http
            scheme: HTTP
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 5
        name: vllm-container
        ports:
        - containerPort: 8000
          name: http
          protocol: TCP
        readinessProbe:
          failureThreshold: 3
          httpGet:
            path: /health
            port: http
            scheme: HTTP
          periodSeconds: 5
          successThreshold: 1
          timeoutSeconds: 5
        resources:
          limits:
            ephemeral-storage: 20Gi
            nvidia.com/gpu: "1"
          requests:
            ephemeral-storage: 20Gi
            nvidia.com/gpu: "1"
        startupProbe:
          failureThreshold: 3600
          httpGet:
            path: /health
            port: http
            scheme: HTTP
          initialDelaySeconds: 5
          periodSeconds: 2
        volumeMounts:
        - mountPath: /dev/shm
          name: dshm
        - mountPath: /data
          name: data
      volumes:
      - emptyDir:
          medium: Memory
          sizeLimit: 128Gi
        name: dshm
      - name: data
# File: ip-manifest.yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: inference-perf
  labels:
    app: inference-perf
spec:
  template:
    metadata:
      labels:
        app: inference-perf
    spec:
      serviceAccountName: inf-perf
      ###serviceAccountName: s3-access
      nodeSelector:
        node.kubernetes.io/instance-type: $CLIENT_MACH
      containers:
        - name: inference-perf
          image: quay.io/inference-perf/inference-perf:v0.4.0
          imagePullPolicy: Always
          command: ["inference-perf"]
          args: ["--config_file", "/etc/config/config.yml", "--log-level", "DEBUG"]
          env:
            - name: HF_TOKEN
              valueFrom:
                secretKeyRef:
```

```
                  name: hf-secret
                  key: hf_api_token
                  optional: true
            volumeMounts:
              - name: config-volume
                mountPath: /etc/config
                readOnly: true
      restartPolicy: Never
      volumes:
        - name: config-volume
          configMap:
            name: inference-perf-config
# File: config.yaml
api:
  type: completion
  streaming: true
data:
  type: shared_prefix
  shared_prefix:
    num_unique_system_prompts: 100
    num_users_per_system_prompt: 10
    system_prompt_len: 1572
    question_len: 64
    output_len: 64
load:
  type: poisson
  stages:
    - rate: 10
      duration: 100
    - rate: 40
      duration: 60
    - rate: 60
      duration: 60
    - rate: 80
      duration: 60
    - rate: 100
      duration: 60
    - rate: 150
      duration: 60
    - rate: 180
      duration: 60
    - rate: 200
      duration: 60
    - rate: 220
      duration: 60
    - rate: 250
      duration: 60
server:
  type: vllm
  model_name: meta-llama/Llama-3.1-8B-Instruct
  base_url: http://${URL}
tokenizer:
  pretrained_model_name_or_path: meta-llama/Llama-3.1-8B-Instruct
  token: ${HF_TOKEN}
metrics:
  type: default
report:
  request_lifecycle:
    summary: true
    per_stage: true
    per_request: false
  prometheus:
    summary: false
    per_stage: false
storage:
  ${BUCKET_TYPE}:
  simple_storage_service:
    bucket_name: cbfgf-bucket
```

This project was commissioned by Google.

Read the report ▶

Primary contributors

Tech: Dan Sullivan

Writing: Laura Weeks

Design: Jared White

PM: Claire Ackerman

How we created this report

A PT team, which includes the contributors we've listed and others, created this report and performed the technical work behind it. We used AI to draft portions of the text.

Principled Technologies®

Facts matter.®