APPENDICES

Appendix 1: Curriculum Vitae

I Gusti Putu Rangga Adithia Putra was born in Denpasar on April 26, 2004. He is an Indonesian citizen and practices Hinduism. He currently resides at Jl. Gunung Guntur, Gang XXIV No. 24, Padangsambian, Denpasar, Bali. He completed his elementary education at SD Negeri 10 Padangsambian. He then continued his education at SMP PGRI 5 Denpasar. After completing junior high school, he pursued vocational education at SMK TI Bali Global Denpasar. He subsequently continued his higher education in the Bachelor's Degree Program in Computer Science at Universitas Pendidikan Ganesha. As part of the requirements for completing his undergraduate studies, he conducted research and prepared a thesis entitled "Comparative Analysis of CPU-Based and Requests-Per-Second (RPS) Metrics in Kubernetes Horizontal Pod Autoscaler (HPA)." During his studies, he developed an interest in computer science, particularly in cloud computing, container orchestration, system performance, and Kubernetes technology. This curriculum vitae was prepared as supplementary information for the completion of his undergraduate thesis.

Appendix 2: Frontend Deployment Config File

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-frontend
  namespace: skripsi-hpa
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-frontend
  template:
    metadata:
      labels:
        app: nginx-frontend
    spec:
      containers:
      - name: nginx
        image: nginx:stable
```

```
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 80
        resources:
          requests:
            cpu: "100m"
            memory: "128Mi"
          limits:
            cpu: "500m"
            memory: "256Mi"
        volumeMounts:
        - name: nginx-main-config
          mountPath: /etc/nginx/nginx.conf
          subPath: nginx.conf

        - name: nginx-server-config
          mountPath: /etc/nginx/conf.d/default.conf
          subPath: default.conf

      - name: nginx-exporter
        image: nginx/nginx-prometheus-exporter:latest
        args:
          - -nginx.scrape-uri=http://127.0.0.1:80/stub_status
        ports:
        - containerPort: 9113
        resources:
          requests:
            cpu: "10m"
            memory: "32Mi"
          limits:
            cpu: "50m"
            memory: "64Mi"
      volumes:
      - name: nginx-main-config
        configMap:
          name: nginx-main-config

      - name: nginx-server-config
        configMap:
          name: nginx-server-config

---
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-main-config
  namespace: skripsi-hpa
data:
  nginx.conf: |
    worker_processes 1;

    events {
        worker_connections 1024;
    }

    http {
        include /etc/nginx/mime.types;
        default_type application/octet-stream;
        include /etc/nginx/conf.d/*.conf;
    }

---
apiVersion: v1
kind: ConfigMap
metadata:
```

```
  name: nginx-server-config
  namespace: skripsi-hpa
data:
  default.conf: |
    server {
        listen 80;

        add_header  Cache-Control  "no-store,  no-cache,  must-revalidate,
proxy-revalidate" always;
        add_header Pragma "no-cache" always;
        add_header Expires "0" always;

        location = /stub_status {
            stub_status;
            access_log off;
            allow 127.0.0.1;
            deny all;
        }

        location / {
            proxy_http_version 1.1;
            proxy_set_header Connection "";
            proxy_pass http://backend;
        }
    }


---
apiVersion: v1
kind: Service
metadata:
  name: nginx-frontend
  namespace: skripsi-hpa
  labels:
    app: nginx-frontend
spec:
  selector:
    app: nginx-frontend
  type: NodePort
  ports:
  - name: http
    port: 80
    targetPort: 80
    nodePort: 30007
  - name: metrics
    port: 9113
    targetPort: 9113
```

Appendix 3: *HPA CPU Config File*

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: nginx-frontend-hpa-cpu
  namespace: skripsi-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: nginx-frontend

  minReplicas: 1
  maxReplicas: 10

  metrics:
```

```
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 41
```

Appendix 4: *HPA RPS Config File*

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: nginx-frontend-hpa-rps
  namespace: skripsi-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: nginx-frontend
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Pods
    pods:
      metric:
        name: nginx_http_requests_per_second
      target:
        type: AverageValue
        averageValue: 80
  behavior:
    scaleUp:
      stabilizationWindowSeconds: 0
      selectPolicy: Max
      policies:
      - type: Percent
        value: 900
        periodSeconds: 15
      - type: Pods
        value: 10
        periodSeconds: 15
```

Appendix 5: *Query Prometheus for getting Pod Replica Frontend*

```
count(
  kube_pod_status_phase{
    namespace="skripsi-hpa",
    pod=~"nginx-frontend.*",
    phase="Running"
  }
)
```

Appendix 6: *Query Prometheus for getting CPU Usage Frontend*

```
avg(
  rate(
    container_cpu_usage_seconds_total{
      namespace="skripsi-hpa",
      pod=~"nginx-frontend.*"
    }[1m]
  )
)
/
0.1
* 100
```

Appendix 7: *Load testing script for Baseline & Ramp-up Load*

```
import http from "k6/http";
import { check, sleep } from "k6";
import { Trend, Rate } from "k6/metrics";

export const responseTime = new Trend("response_time", true);
export const successRate = new Rate("success_rate");

export const options = {
  scenarios: {
    baseline_rps: {
      executor: "ramping-arrival-rate",
      startRate: 50,
      timeUnit: "1s",
      preAllocatedVUs: 200,
      maxVUs: 500,
      stages: [
        { target: 50, duration: "1m" },
        { target: 100, duration: "1m" },
        { target: 150, duration: "1m" },
        { target: 200, duration: "1m" },
        { target: 250, duration: "1m" },
        { target: 300, duration: "1m" },
        { target: 350, duration: "1m" },
        { target: 400, duration: "1m" },
        { target: 450, duration: "1m" },
        { target: 500, duration: "1m" },
      ],
    },
  },
};

export default function () {
  const res = http.get("http://target_url/", {
    tags: {
      scenario: "baseline_rps",
    },
  });

  const ok = check(res, {
    "status is 200": (r) => r.status === 200,
  });

  responseTime.add(res.timings.duration, {
    status: String(res.status)
  });
  successRate.add(ok);
}
```

Appendix 8: *Load testing script for Cyclic Load*

```
import http from "k6/http";
import { check } from "k6";
import { Trend, Rate } from "k6/metrics";

export const responseTime = new Trend("response_time", true);
export const successRate = new Rate("success_rate");

export const options = {
  scenarios: {
    cyclic_rps: {
      executor: "ramping-arrival-rate",
      startRate: 50,
      timeUnit: "1s",
```

```
      preAllocatedVUs: 200,
      maxVUs: 1000,
      stages: [
        { target: 500, duration: "30s" },
        { target: 500, duration: "2m" },
        { target: 50, duration: "30s" },
        { target: 50, duration: "2m" },
        { target: 500, duration: "30s" },
        { target: 500, duration: "2m" },
        { target: 50, duration: "30s" },
        { target: 50, duration: "2m" },
      ],
    },
  },
};

export default function () {
  const res = http.get("http://target_url", {
    tags: {
      scenario: "cyclic_rps",
    },
  });

  const ok = check(res, {
    "status is 200": (r) => r.status === 200,
  });

  responseTime.add(res.timings.duration, {
    status: String(res.status)
  });
  successRate.add(ok);
}
```

Appendix 9: *Load testing script for Sustained High Load*

```
import http from "k6/http";
import { check } from "k6";
import { Trend, Rate } from "k6/metrics";

export const responseTime = new Trend("response_time", true);
export const successRate = new Rate("success_rate");

export const options = {
  scenarios: {
    high_sustain_load: {
      executor: "ramping-arrival-rate",
      startRate: 50,
      timeUnit: "1s",
      preAllocatedVUs: 300,
      maxVUs: 1000,
      stages: [
        { target: 50, duration: "1m" },
        { target: 500, duration: "1m" },
        { target: 500, duration: "1m" },
        { target: 500, duration: "1m" },
        { target: 500, duration: "1m" },
        { target: 500, duration: "1m" },
        { target: 500, duration: "1m" },
        { target: 500, duration: "1m" },
        { target: 500, duration: "1m" },
        { target: 500, duration: "1m" },
      ],
    },
```

```
  },
};

export default function () {
  const res = http.get("http://target_url/", {
    tags: {
      scenario: "high_sustain_load",
    },
  });

  const ok = check(res, {
    "status is 200": (r) => r.status === 200,
  });

  responseTime.add(res.timings.duration, {
    status: String(res.status),
  });

  successRate.add(ok);
}
```

Appendix 10: *Load testing script for Gradual Decrease Load*

```
import http from "k6/http";
import { check } from "k6";
import { Trend, Rate } from "k6/metrics";

export const responseTime = new Trend("response_time", true);
export const successRate = new Rate("success_rate");

export const options = {
  scenarios: {
    gradual_scaledown_rps: {
      executor: "ramping-arrival-rate",
      startRate: 500,
      timeUnit: "1s",
      preAllocatedVUs: 500,
      maxVUs: 1000,
      stages: [
        { target: 500, duration: "1m" },
        { target: 450, duration: "1m" },
        { target: 400, duration: "1m" },
        { target: 350, duration: "1m" },
        { target: 300, duration: "1m" },
        { target: 250, duration: "1m" },
        { target: 200, duration: "1m" },
        { target: 150, duration: "1m" },
        { target: 100, duration: "1m" },
        { target: 50, duration: "1m" },
      ],
    },
  },
};

export default function () {
  const res = http.get("http://target_url/", {
    tags: {
      scenario: "gradual_scaledown_rps",
    },
  });

  const ok = check(res, {
    "status is 200": (r) => r.status === 200,
  });
```

```
  responseTime.add(res.timings.duration, {
    status: String(res.status),
  });
  successRate.add(ok);
}
```

Appendix 11: *Load Testing Log*

https://drive.google.com/drive/folders/1Qo_D4C8Ham0jOKfLlJxQYqgIv8hFA-Qo?usp=sharing