Hands-on Guide: Using Valkey with Amazon ElastiCache Serverless

Table of Contents

Introduction

As demand for high-performance caching and messaging systems grows, developers and businesses are exploring open-source alternatives to Redis. One such emerging solution is Valkey, a Redis-compatible, community-driven, open-source key-value store.

This guide is a comprehensive reference for:

- Deploying Valkey using Amazon ElastiCache Serverless
- Performing performance benchmarking
- Seamlessly upgrading from Redis OSS to Valkey

What is Valkey?

Valkey is a high-performance, in-memory key-value data store designed for fast, scalable, and efficient data operations. It maintains Redis protocol compatibility, allowing it to act as a drop-in replacement.

Core Use Cases:

- In-memory data caching
- Session management
- Message queues
- Real-time analytics
- Primary NoSQL database for low-latency applications

Community and Licensing:

- Backed by the Linux Foundation
- BSD-licensed
- 21,000+ GitHub stars
- 800+ contributors
- Actively maintained and improved

Why Migrate from Redis to Valkey?

Valkey is quickly becoming the preferred Redis alternative due to the following reasons:

1. Licensing Flexibility

Redis transitioned from an open-source license to AGPL, introducing restrictions for commercial use. Valkey remains BSD-licensed, making it safe and flexible for enterprise deployment.

2. Open Governance

Valkey is governed by an open community under the Linux Foundation, ensuring transparent development and long-term reliability.

3. Performance Enhancements

Valkey delivers improved throughput and reduced latency over Redis OSS. Optimizations include lower CPU usage and memory overhead.

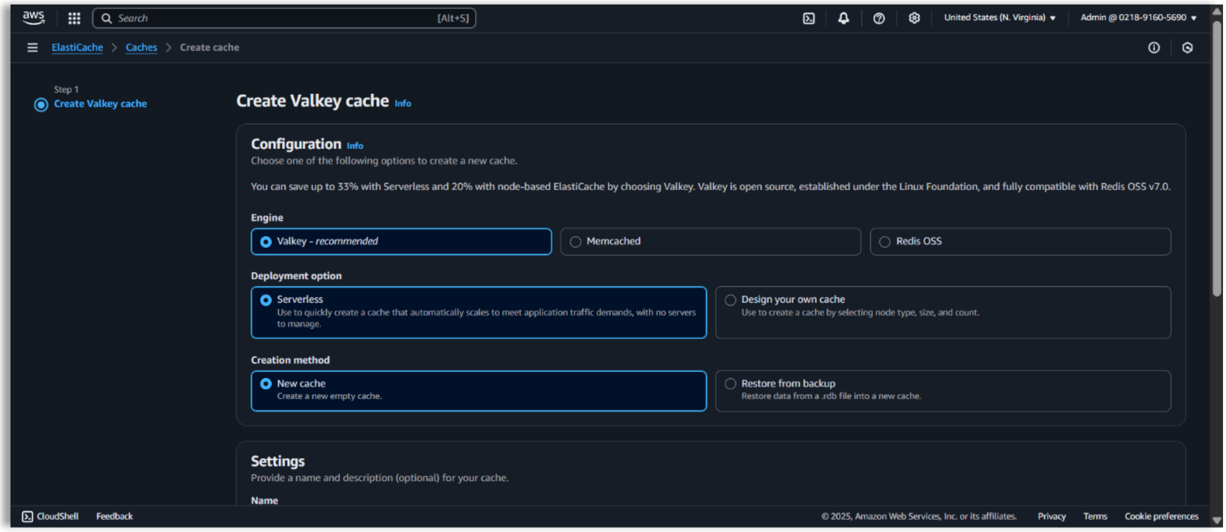
4. Compatibility

Valkey supports the same protocol and commands as Redis, making it a drop-in replacement with minimal application-level changes.

Setting Up Valkey with Amazon ElastiCache Serverless

Amazon provides Valkey as a serverless managed option in ElastiCache, allowing developers to focus on application logic instead of infrastructure.

Setup Steps:



Create Valkey cache

Configuration

Choose one of the following options to create a new cache.

You can save up to 33% with Serverless and 20% with node-based ElastiCache by choosing Valkey. Valkey is open source, established under the Linux Foundation, and fully compatible with Redis OSS v7.0.

Engine

● Valkey - *recommended* ○ Memcached ○ Redis OSS

Deployment option

● Serverless
Use to quickly create a cache that automatically scales to meet application traffic demands, with no servers to manage.

○ Design your own cache
Use to create a cache by selecting node type, size, and count.

Creation method

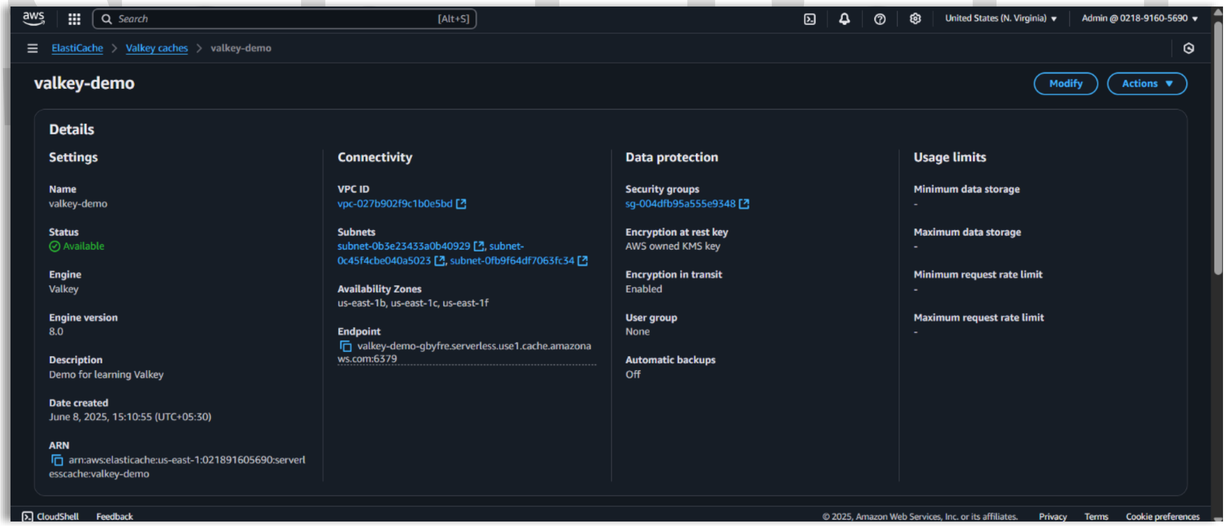
● New cache
Create a new empty cache.

○ Restore from backup
Restore data from a .rdb file into a new cache.

Settings

Provide a name and description (optional) for your cache.

Name


aws
Search
[Alt+S]
United States (N. Virginia) ▾
Admin @ 0218-9160-5690 ▾
ElastiCache > Valkey caches > valkey-demo
Modify
Actions ▼
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

valkey-demo

Details

Settings

Name
valkey-demo

Status
Available

Engine
Valkey

Engine version
8.0

Description
Demo for learning Valkey

Date created
June 8, 2025, 15:10:55 (UTC+05:30)

ARN
arn:aws:elasticache:us-east-1:021891605690:serverlesscache:valkey-demo

Connectivity

VPC ID
vpc-027b902f9c1b0e5bd

Subnets
subnet-0b3e23433a0b40929, subnet-0c45f4cbe040a5023, subnet-0fb9f64df7063fc34

Availability Zones
us-east-1b, us-east-1c, us-east-1f

Endpoint
valkey-demo-gbyfre.serverless.use1.cache.amazonaws.com:6379

Data protection

Security groups
sg-004dfb95a555e9348

Encryption at rest key
AWS owned KMS key

Encryption in transit
Enabled

User group
None

Automatic backups
Off

Usage limits

Minimum data storage
-

Maximum data storage
-

Minimum request rate limit
-

Maximum request rate limit
-

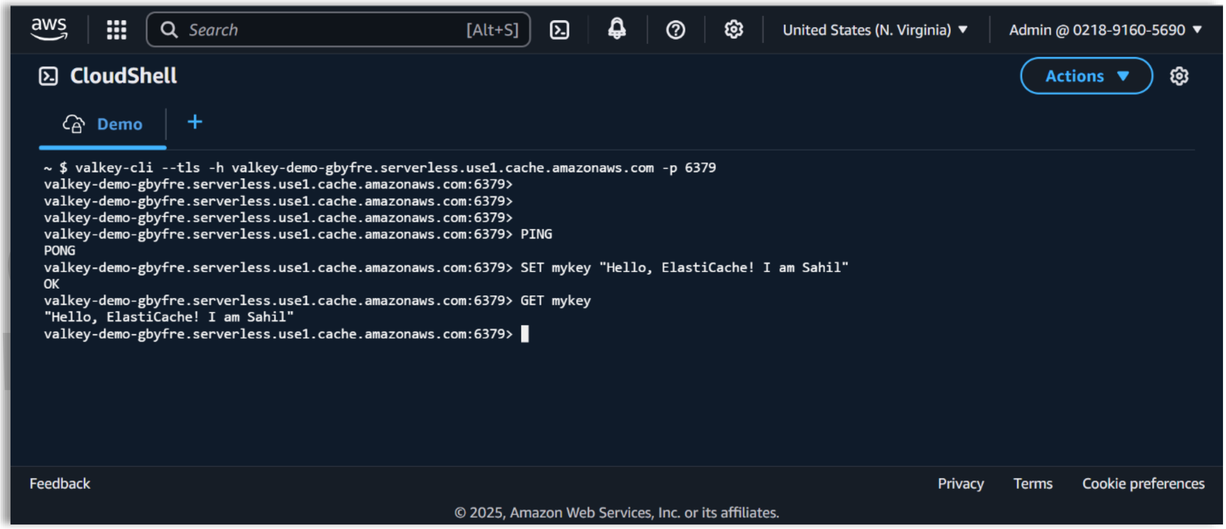
1. Log in to the AWS Console
2. Navigate to Amazon ElastiCache
3. Click "Get Started"
4. Choose "Valkey" as the engine
5. Select "Serverless" deployment

6. Fill in configuration:
    - Name: valkey-demo
    - Engine Version: 8 (recommended)
7. Adjust advanced settings (VPC, encryption, etc.)
8. Click "Create"

In a few minutes, your cache becomes active and provides an endpoint for connectivity.

Connecting to Valkey

Using AWS CloudShell:



```
~ $ valkey-cli --tls -h valkey-demo-gbyfre.serverless.use1.cache.amazonaws.com -p 6379
valkey-demo-gbyfre.serverless.use1.cache.amazonaws.com:6379>
valkey-demo-gbyfre.serverless.use1.cache.amazonaws.com:6379>
valkey-demo-gbyfre.serverless.use1.cache.amazonaws.com:6379>
valkey-demo-gbyfre.serverless.use1.cache.amazonaws.com:6379> PING
PONG
valkey-demo-gbyfre.serverless.use1.cache.amazonaws.com:6379> SET mykey "Hello, ElastiCache! I am Sahil"
OK
valkey-demo-gbyfre.serverless.use1.cache.amazonaws.com:6379> GET mykey
"Hello, ElastiCache! I am Sahil"
valkey-demo-gbyfre.serverless.use1.cache.amazonaws.com:6379>
```

1. From the cache instance page, go to "Connectivity & Security"
2. Click "Connect to cache" to launch AWS CloudShell
3. Run the following command to connect:
4. ./valkey-cli -h <your-endpoint> -p 6379
5. Test with basic commands:
6. set greeting "Hello, Valkey"
7. get greeting

Expected output:

"Hello, Valkey"

Programmatic Access Using Valkey Clients

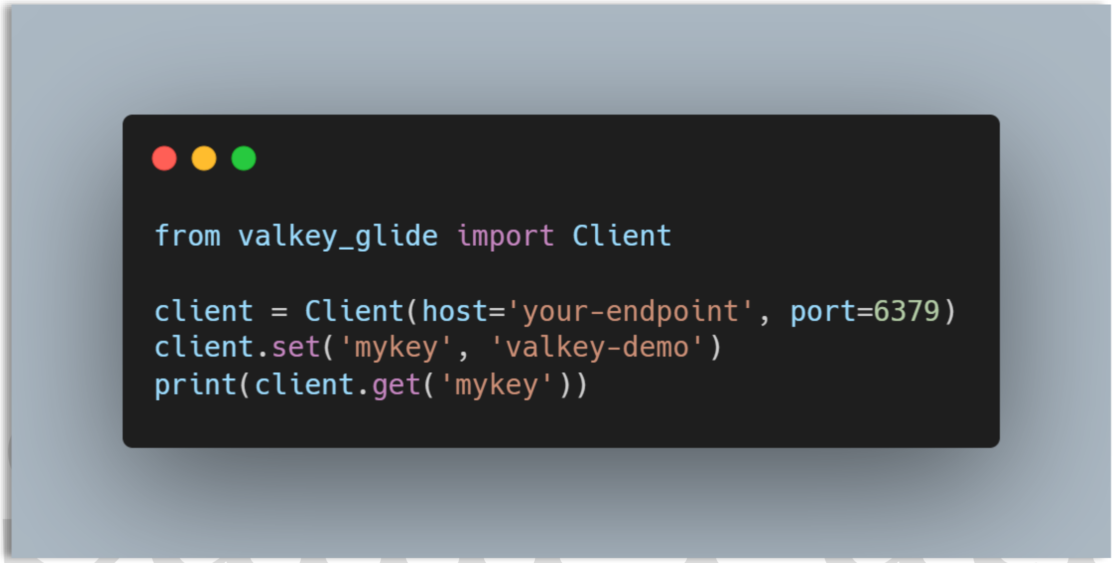
You can integrate Valkey into your applications using Valkey Glide or other Redis-compatible libraries.

Python Example:

Install the client:

pip install valkey-glide

Sample script:



```python
from valkey_glide import Client

client = Client(host='your-endpoint', port=6379)
client.set('mykey', 'valkey-demo')
print(client.get('mykey'))
```

Valkey also works with existing Redis libraries like redis-py, ioredis, or Jedis.

Benchmarking Valkey vs Redis OSS

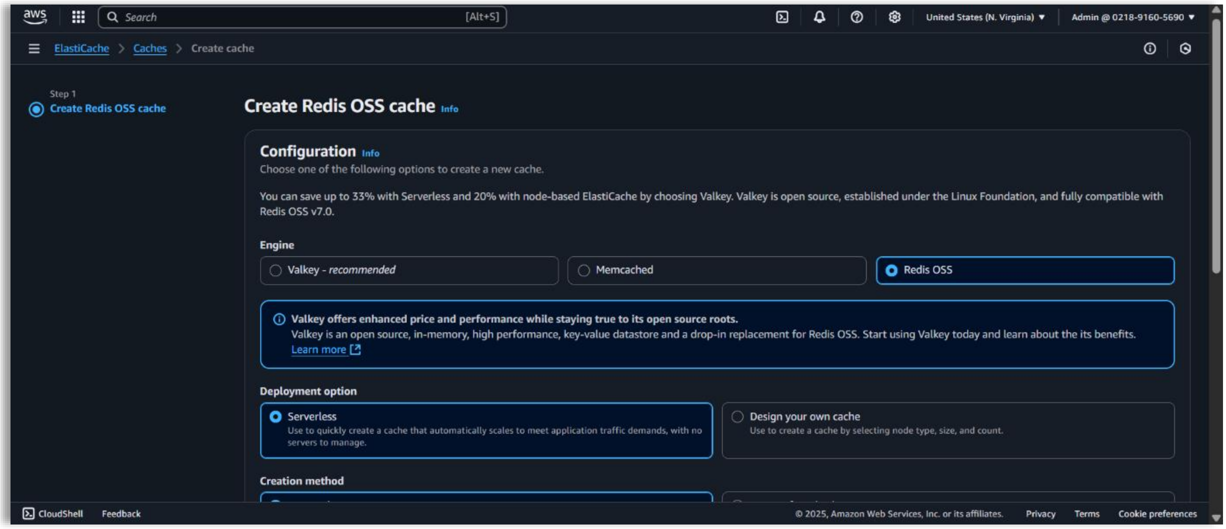
Objective:

Evaluate performance across key metrics:

- Operations per second (throughput)
- Latency (P50, P90, P99)

Setup:

- Two ElastiCache instances: one Valkey, one Redis OSS, EC2 instance to run benchmarking script, Python script performing set, get, and delete operations


ElastiCache > Caches > Create cache

Step 1
Create Redis OSS cache

Create Redis OSS cache

Configuration
Choose one of the following options to create a new cache.

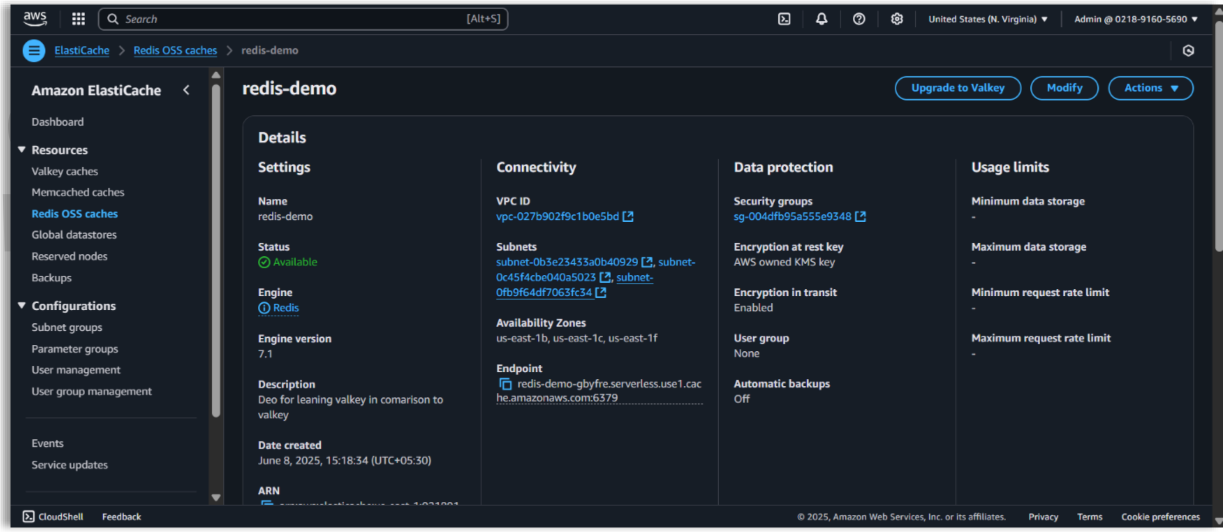
You can save up to 33% with Serverless and 20% with node-based ElastiCache by choosing Valkey. Valkey is open source, established under the Linux Foundation, and fully compatible with Redis OSS v7.0.

Engine: ○ Valkey - *recommended* ○ Memcached ● Redis OSS

Valkey offers enhanced price and performance while staying true to its open source roots.
Valkey is an open source, in-memory, high performance, key-value datastore and a drop-in replacement for Redis OSS. Start using Valkey today and learn about the its benefits.
Learn more

Deployment option:
● Serverless — Use to quickly create a cache that automatically scales to meet application traffic demands, with no servers to manage.
○ Design your own cache — Use to create a cache by selecting node type, size, and count.

Creation method


ElastiCache > Redis OSS caches > redis-demo

redis-demo

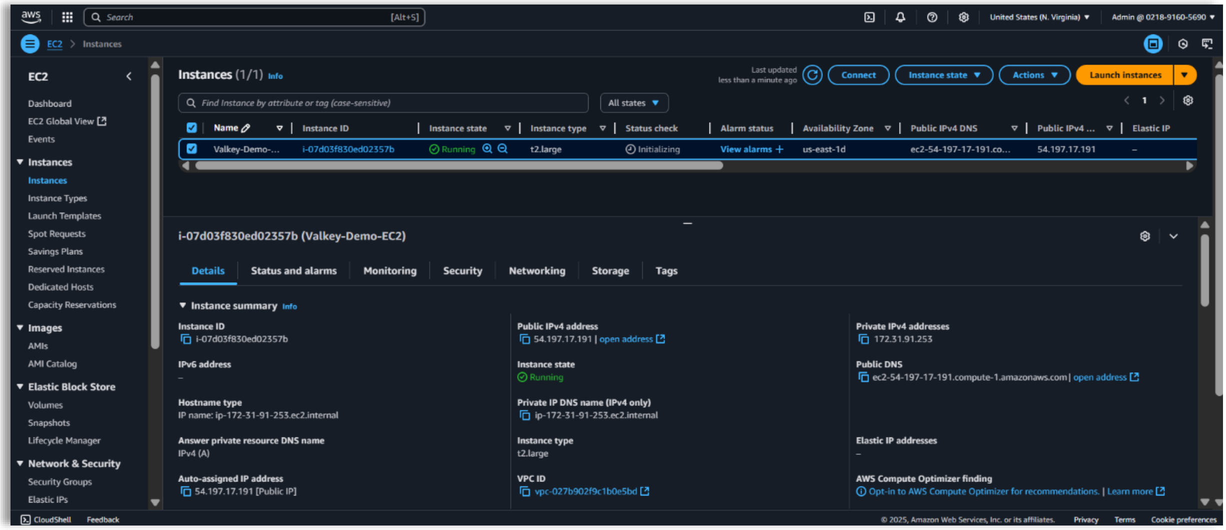
Details

Settings
Name: redis-demo
Status: Available
Engine: Redis
Engine version: 7.1
Description: Deo for leaning valkey in comarison to valkey
Date created: June 8, 2025, 15:18:34 (UTC+05:30)
ARN

Connectivity
VPC ID: vpc-027b902f9c1b0e5bd
Subnets: subnet-0b3e23433a0b40929, subnet-0c45f4cbe040a5023, subnet-0fb9f64df7063fc34
Availability Zones: us-east-1b, us-east-1c, us-east-1f
Endpoint: redis-demo-gbyfre.serverless.use1.cache.amazonaws.com:6379

Data protection
Security groups: sg-004dfb95a555e9348
Encryption at rest key: AWS owned KMS key
Encryption in transit: Enabled
User group: None
Automatic backups: Off

Usage limits
Minimum data storage: -
Maximum data storage: -
Minimum request rate limit: -
Maximum request rate limit: -


EC2 > Instances

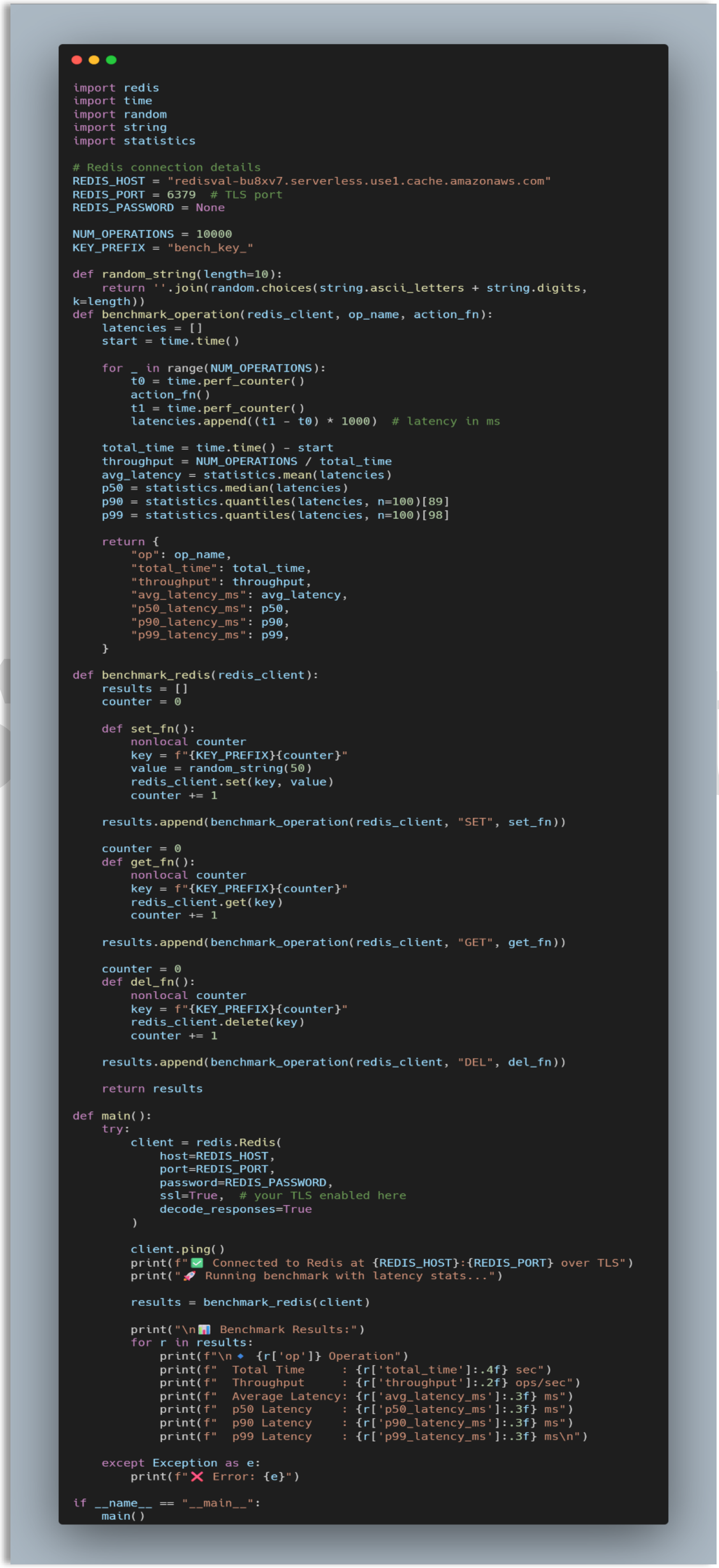
Instances (1/1)

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
Valkey-Demo-...	i-07d03f830ed02357b	Running	t2.large	Initializing	View alarms +	us-east-1d	ec2-54-197-17-191.co...	54.197.17.191	–

i-07d03f830ed02357b (Valkey-Demo-EC2)

Instance summary

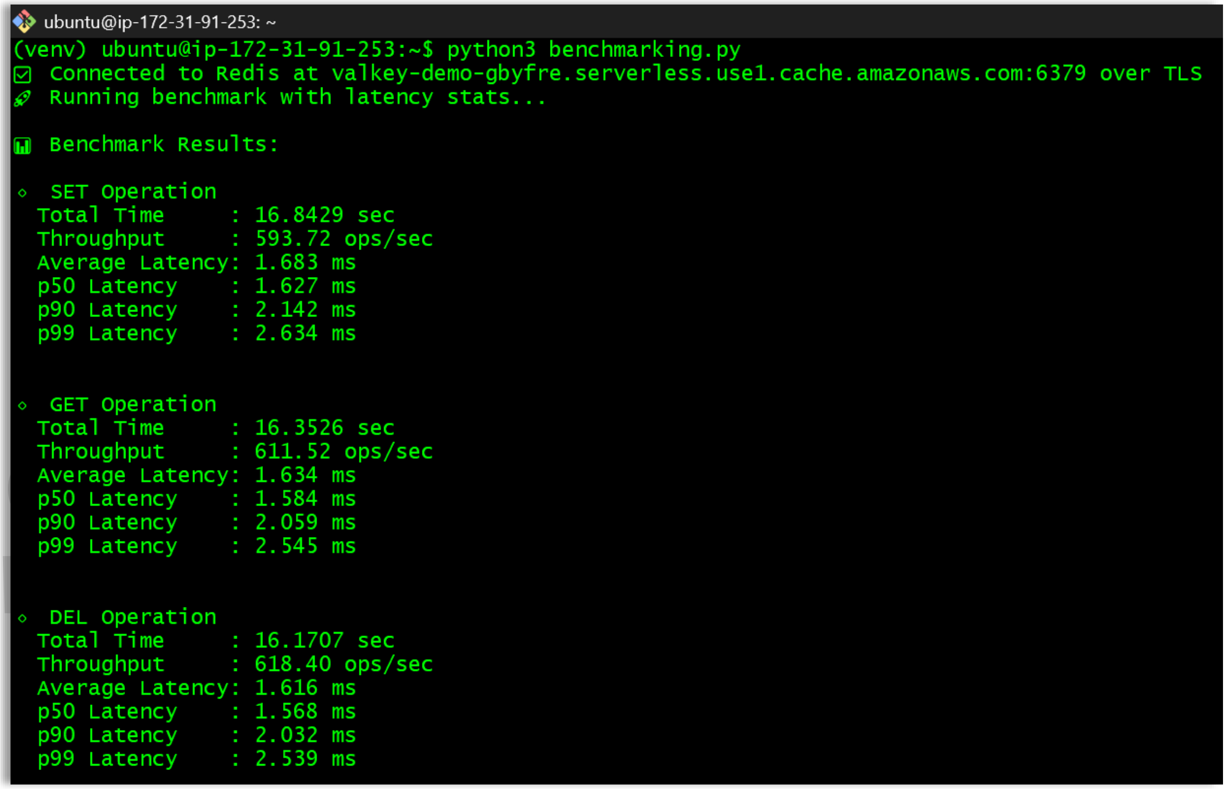
Instance ID: i-07d03f830ed02357b
IPv6 address: –
Hostname type: IP name: ip-172-31-91-253.ec2.internal
Answer private resource DNS name: IPv4 (A)
Auto-assigned IP address: 54.197.17.191 [Public IP]
Public IPv4 address: 54.197.17.191 | open address
Instance state: Running
Private IP DNS name (IPv4 only): ip-172-31-91-253.ec2.internal
Instance type: t2.large
VPC ID: vpc-027b902f9c1b0e5bd
Private IPv4 addresses: 172.31.91.253
Public DNS: ec2-54-197-17-191.compute-1.amazonaws.com | open address
Elastic IP addresses: –
AWS Compute Optimizer finding: Opt-in to AWS Compute Optimizer for recommendations. | Learn more



```python
import redis
import time
import random
import string
import statistics

# Redis connection details
REDIS_HOST = "redisval-bu8xv7.serverless.use1.cache.amazonaws.com"
REDIS_PORT = 6379  # TLS port
REDIS_PASSWORD = None

NUM_OPERATIONS = 10000
KEY_PREFIX = "bench_key_"

def random_string(length=10):
    return ''.join(random.choices(string.ascii_letters + string.digits,
k=length))
def benchmark_operation(redis_client, op_name, action_fn):
    latencies = []
    start = time.time()

    for _ in range(NUM_OPERATIONS):
        t0 = time.perf_counter()
        action_fn()
        t1 = time.perf_counter()
        latencies.append((t1 - t0) * 1000)  # latency in ms

    total_time = time.time() - start
    throughput = NUM_OPERATIONS / total_time
    avg_latency = statistics.mean(latencies)
    p50 = statistics.median(latencies)
    p90 = statistics.quantiles(latencies, n=100)[89]
    p99 = statistics.quantiles(latencies, n=100)[98]

    return {
        "op": op_name,
        "total_time": total_time,
        "throughput": throughput,
        "avg_latency_ms": avg_latency,
        "p50_latency_ms": p50,
        "p90_latency_ms": p90,
        "p99_latency_ms": p99,
    }

def benchmark_redis(redis_client):
    results = []
    counter = 0

    def set_fn():
        nonlocal counter
        key = f"{KEY_PREFIX}{counter}"
        value = random_string(50)
        redis_client.set(key, value)
        counter += 1

    results.append(benchmark_operation(redis_client, "SET", set_fn))

    counter = 0
    def get_fn():
        nonlocal counter
        key = f"{KEY_PREFIX}{counter}"
        redis_client.get(key)
        counter += 1

    results.append(benchmark_operation(redis_client, "GET", get_fn))

    counter = 0
    def del_fn():
        nonlocal counter
        key = f"{KEY_PREFIX}{counter}"
        redis_client.delete(key)
        counter += 1

    results.append(benchmark_operation(redis_client, "DEL", del_fn))

    return results

def main():
    try:
        client = redis.Redis(
            host=REDIS_HOST,
            port=REDIS_PORT,
            password=REDIS_PASSWORD,
            ssl=True,  # your TLS enabled here
            decode_responses=True
        )

        client.ping()
        print(f"✅ Connected to Redis at {REDIS_HOST}:{REDIS_PORT} over TLS")
        print("🚀 Running benchmark with latency stats...")

        results = benchmark_redis(client)

        print("\n📊 Benchmark Results:")
        for r in results:
            print(f"\n🔹 {r['op']} Operation")
            print(f"  Total Time     : {r['total_time']:.4f} sec")
            print(f"  Throughput     : {r['throughput']:.2f} ops/sec")
            print(f"  Average Latency: {r['avg_latency_ms']:.3f} ms")
            print(f"  p50 Latency    : {r['p50_latency_ms']:.3f} ms")
            print(f"  p90 Latency    : {r['p90_latency_ms']:.3f} ms")
            print(f"  p99 Latency    : {r['p99_latency_ms']:.3f} ms\n")

    except Exception as e:
        print(f"❌ Error: {e}")

if __name__ == "__main__":
    main()
```

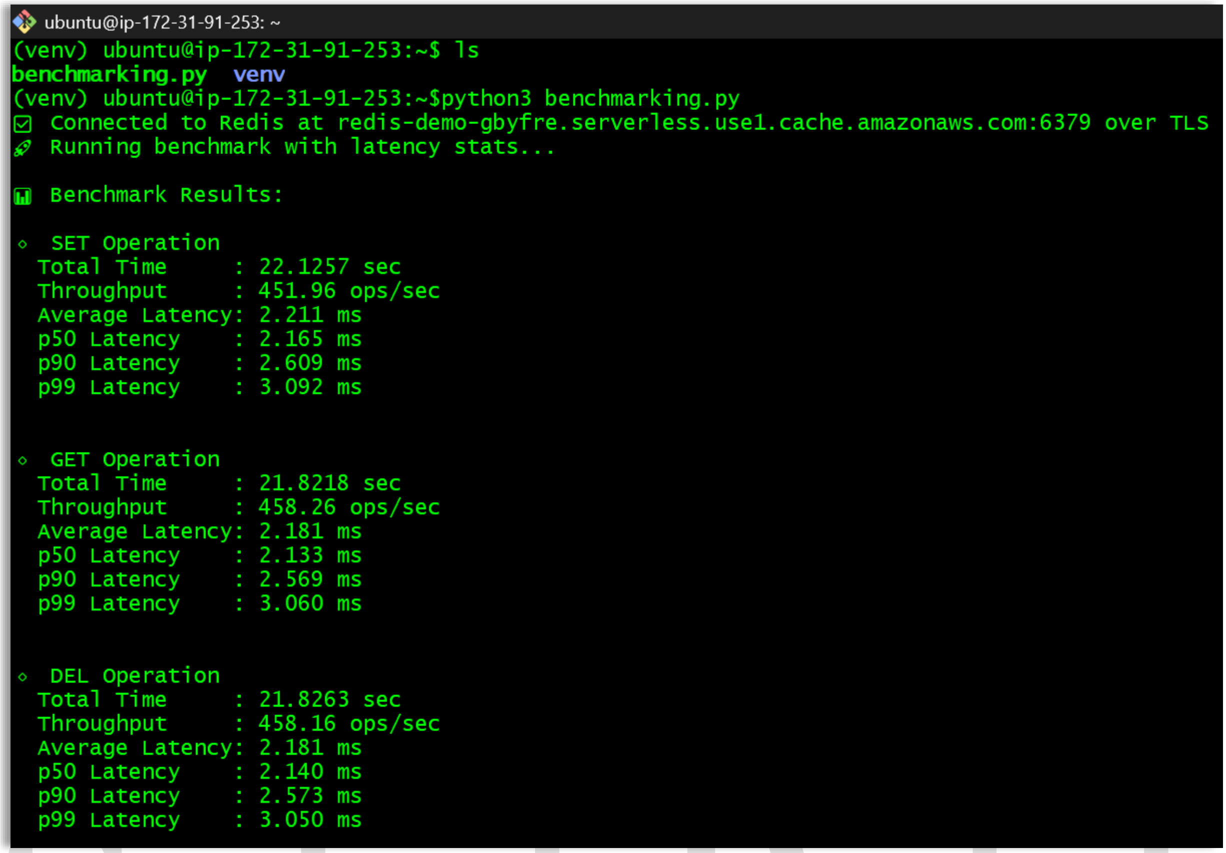
1. Edit benchmarking.py to include your cache endpoint.
2. Run: python3 benchmarking.py

Observations:



ubuntu@ip-172-31-91-253: ~

```
(venv) ubuntu@ip-172-31-91-253:~$ python3 benchmarking.py
☑ Connected to Redis at valkey-demo-gbyfre.serverless.use1.cache.amazonaws.com:6379 over TLS
🚀 Running benchmark with latency stats...

📊 Benchmark Results:

◇ SET Operation
  Total Time      : 16.8429 sec
  Throughput      : 593.72 ops/sec
  Average Latency: 1.683 ms
  p50 Latency     : 1.627 ms
  p90 Latency     : 2.142 ms
  p99 Latency     : 2.634 ms


◇ GET Operation
  Total Time      : 16.3526 sec
  Throughput      : 611.52 ops/sec
  Average Latency: 1.634 ms
  p50 Latency     : 1.584 ms
  p90 Latency     : 2.059 ms
  p99 Latency     : 2.545 ms


◇ DEL Operation
  Total Time      : 16.1707 sec
  Throughput      : 618.40 ops/sec
  Average Latency: 1.616 ms
  p50 Latency     : 1.568 ms
  p90 Latency     : 2.032 ms
  p99 Latency     : 2.539 ms
```



```
ubuntu@ip-172-31-91-253: ~
(venv) ubuntu@ip-172-31-91-253:~$ ls
benchmarking.py  venv
(venv) ubuntu@ip-172-31-91-253:~$python3 benchmarking.py
☑ Connected to Redis at redis-demo-gbyfre.serverless.use1.cache.amazonaws.com:6379 over TLS
🚀 Running benchmark with latency stats...

📊 Benchmark Results:

◇  SET Operation
  Total Time     : 22.1257 sec
  Throughput     : 451.96 ops/sec
  Average Latency: 2.211 ms
  p50 Latency    : 2.165 ms
  p90 Latency    : 2.609 ms
  p99 Latency    : 3.092 ms


◇  GET Operation
  Total Time     : 21.8218 sec
  Throughput     : 458.26 ops/sec
  Average Latency: 2.181 ms
  p50 Latency    : 2.133 ms
  p90 Latency    : 2.569 ms
  p99 Latency    : 3.060 ms


◇  DEL Operation
  Total Time     : 21.8263 sec
  Throughput     : 458.16 ops/sec
  Average Latency: 2.181 ms
  p50 Latency    : 2.140 ms
  p90 Latency    : 2.573 ms
  p99 Latency    : 3.050 ms
```

- Valkey consistently showed lower latency at all percentiles
- Throughput was higher in all tested workloads
- Especially strong in high-concurrency and read-heavy scenarios

---

Seamless Upgrade from Redis OSS to Valkey

Amazon ElastiCache Serverless allows one-click upgrades from Redis OSS to Valkey.

Upgrade Steps:

1. Open your Redis OSS cache in ElastiCache
2. Click "Upgrade to Valkey"
3. Select Engine Version 8
4. Confirm and proceed

Upgrade completes in minutes with no data loss or service downtime.

Summary

Feature	**Redis OSS**	**Valkey**
License	AGPL	BSD
Governance	Redis Ltd	Linux Foundation
Compatibility	Native Redis	Redis-Compatible
Performance	Good	Better
Cloud Support	Yes	Yes
Upgrade Path	Manual	One-click in AWS

Best Practices

- Benchmark workloads before and after migration
- Enable encryption in transit and at rest for production use
- Choose Valkey Glide for advanced client features
- Test upgrades in a non-production environment
- Use CloudWatch for metrics and alerting
- Leverage serverless deployment for cost optimization