In case AWS UAT environments are standalone and not exposed to external networks, and you don't have load balancing enabled, then one can consider using an embedded Redis server for your E2E tests. Here's a recommended approach for setting up embedded Redis in such an environment:

1. **Use Docker**: Containerization with Docker is a convenient way to run an embedded Redis server alongside your application. Docker allows you to create isolated environments for your application and its dependencies.
2. **Docker Compose**: Create a Docker Compose configuration that defines both your application and the Redis container. This allows you to define the services and their interdependencies in a single configuration file.
3. **Volume Mounts**: Configure Docker Compose to use volume mounts for data persistence. This ensures that data is retained between container restarts. You can mount a directory on the host machine to store Redis data files.
4. **Networking**: Ensure that the application container can communicate with the Redis container using Docker's internal network. By default, containers within the same Docker Compose stack can communicate with each other via service names.

Here's an example **docker-compose.yml** file:

yaml

version: '3' services: myapp: image: my-app-image ports: - "8080:8080" depends\_on: - redis volumes: - ./app-data:/app/data networks: - myapp-network redis: image: redis:latest volumes: - ./redis-data:/data networks: - myapp-network networks: myapp-network:

In this example:

* The **myapp** service represents your application.
* The **redis** service represents the embedded Redis server using the official Redis Docker image.
* Both services use volume mounts to persist data (**./app-data** for the application and **./redis-data** for Redis).

By using Docker Compose, you can manage the lifecycle of both your application and the embedded Redis server, ensuring they start and stop together. This approach provides data persistence and network isolation for your tests.

Remember to configure your application to connect to the Redis server using the hostname **redis** (the service name defined in the Docker Compose file) and the appropriate port (usually the default Redis port 6379).

With this setup, you can run your E2E tests in an isolated environment, and the embedded Redis server will be available for your application during testing.