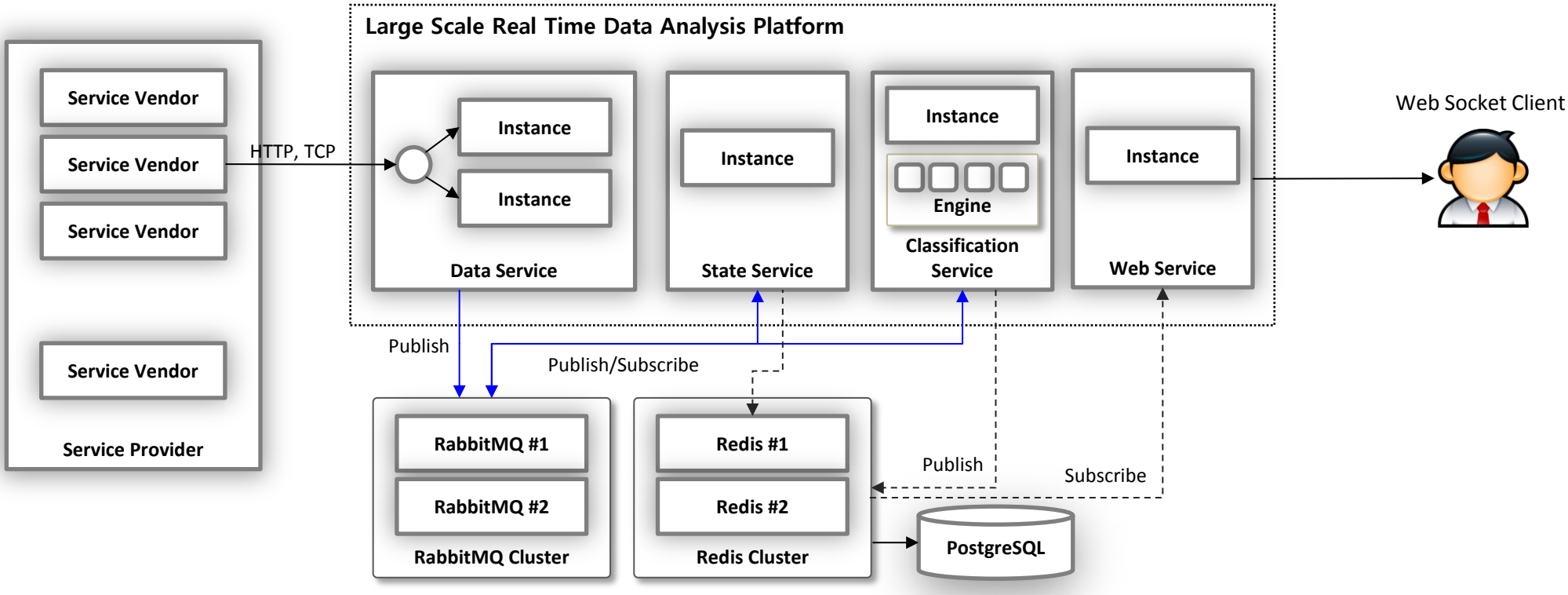
Real Time Analysis

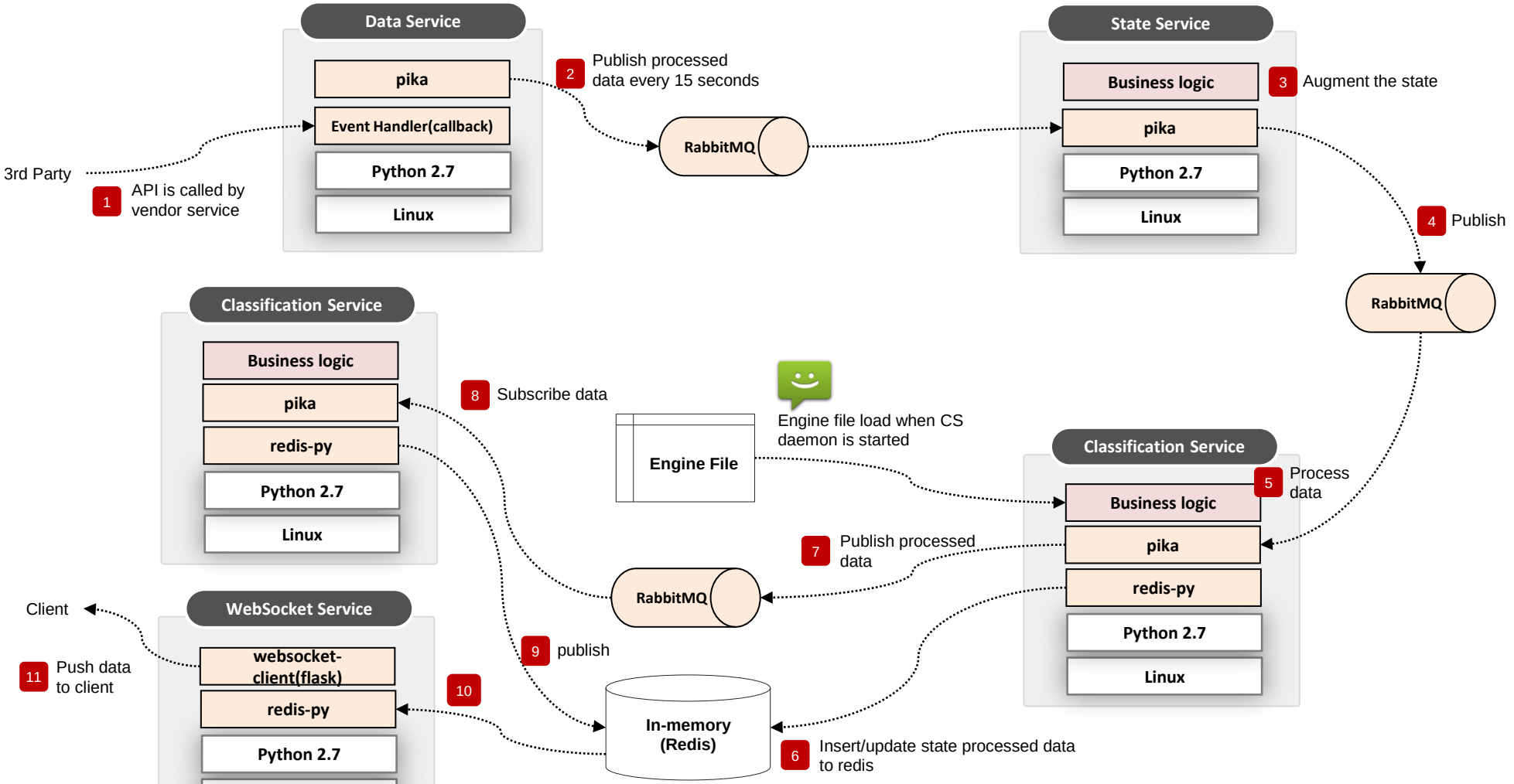
Open Source Consulting, Inc

System Architecture



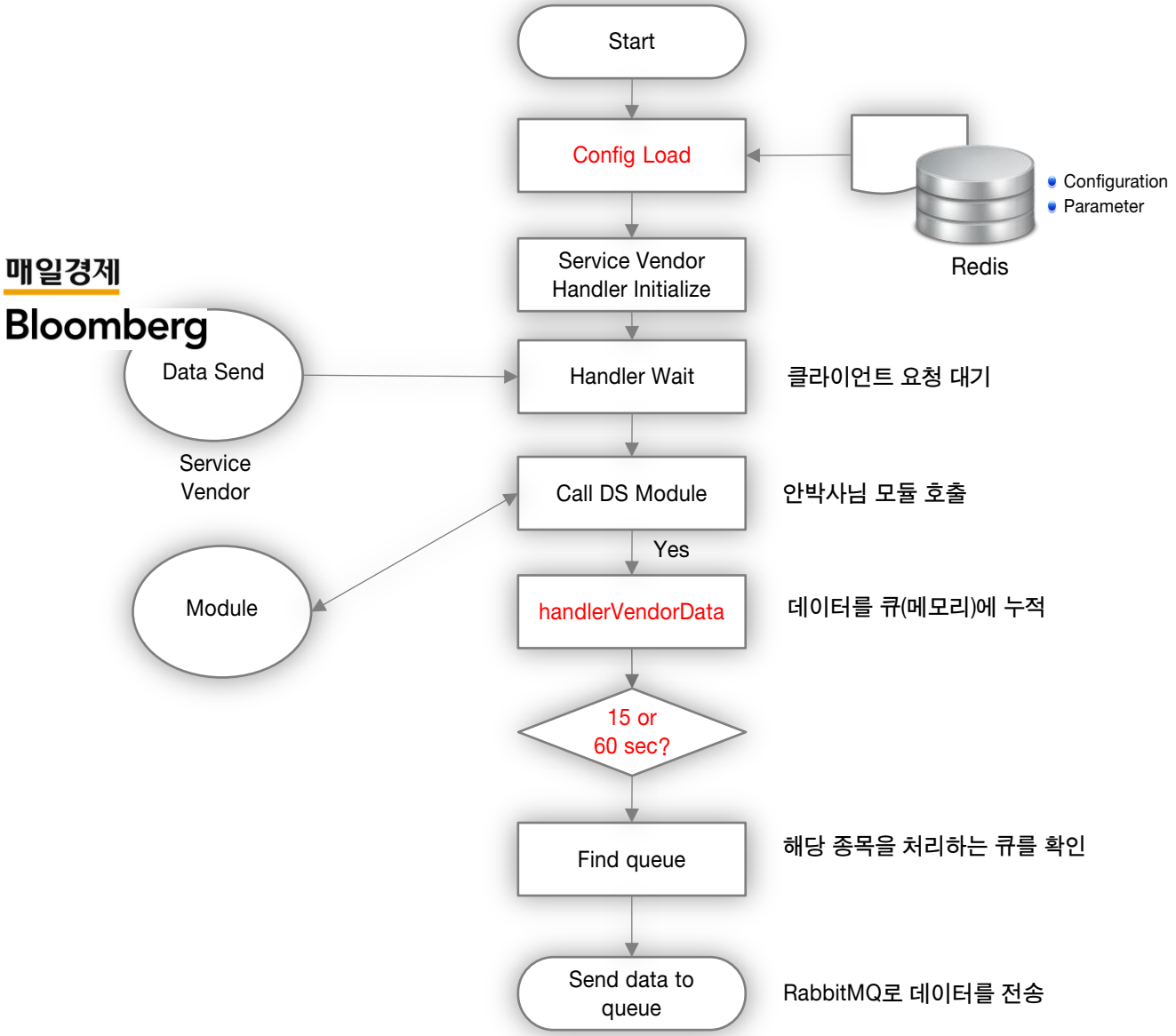
Data Service	• Client로부터 입력되는 서비스는 TCP 소켓과 HTTP 등의 서비스 벤더 규약에 따라 처리 • 종목 5,000개를 기준으로 1분 단위의 message publish
Classification Service	• 서비스 시작 시 300M의 처리 엔진을 메모리로 로딩 • SS가 publish한 데이터를 subscribe • 상태 분류 작업 및 캐시(Redis) 업데이트
State Service	• DS가 publish한 데이터를 subscribe - Order guarantee • 종목에 대한 상태 추가 후 message publish
Web Service	• 사전에 연결된 웹 소켓을 통해 클라이언트에 연결 • 캐시로 들어오는 메시지를 인식하고 클라이언트 푸시 진행 • Flask를 활용하여 웹 서비스를 구축

Data Flow

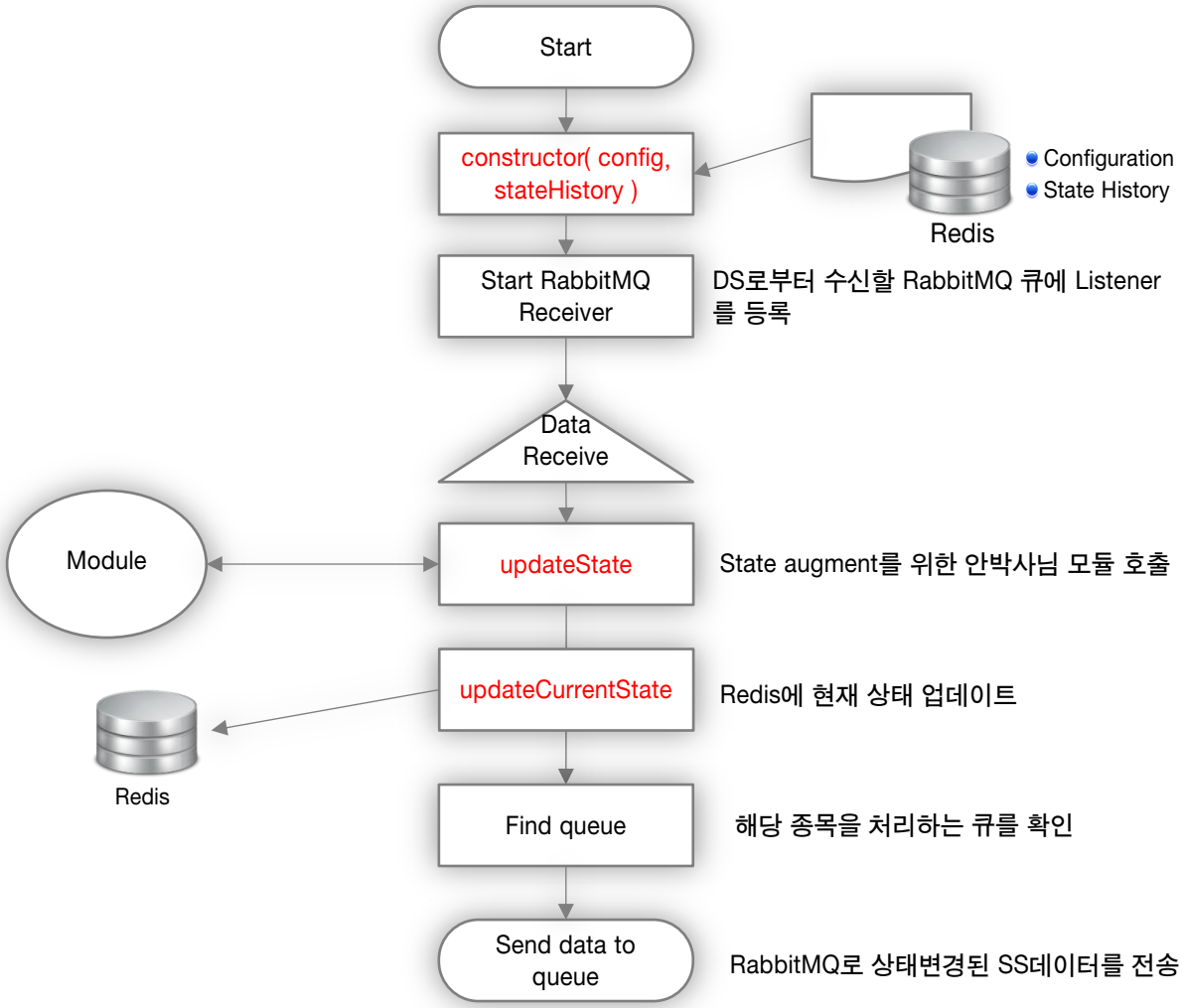


- 모든 component (DS, SS, CS) 가 Redis 에서 config, state (state history), parameter 등을 받아 service 를 시작
- DS: Config + Parameter
- SS: Config + State History
- CS: Config + Engine Serialization

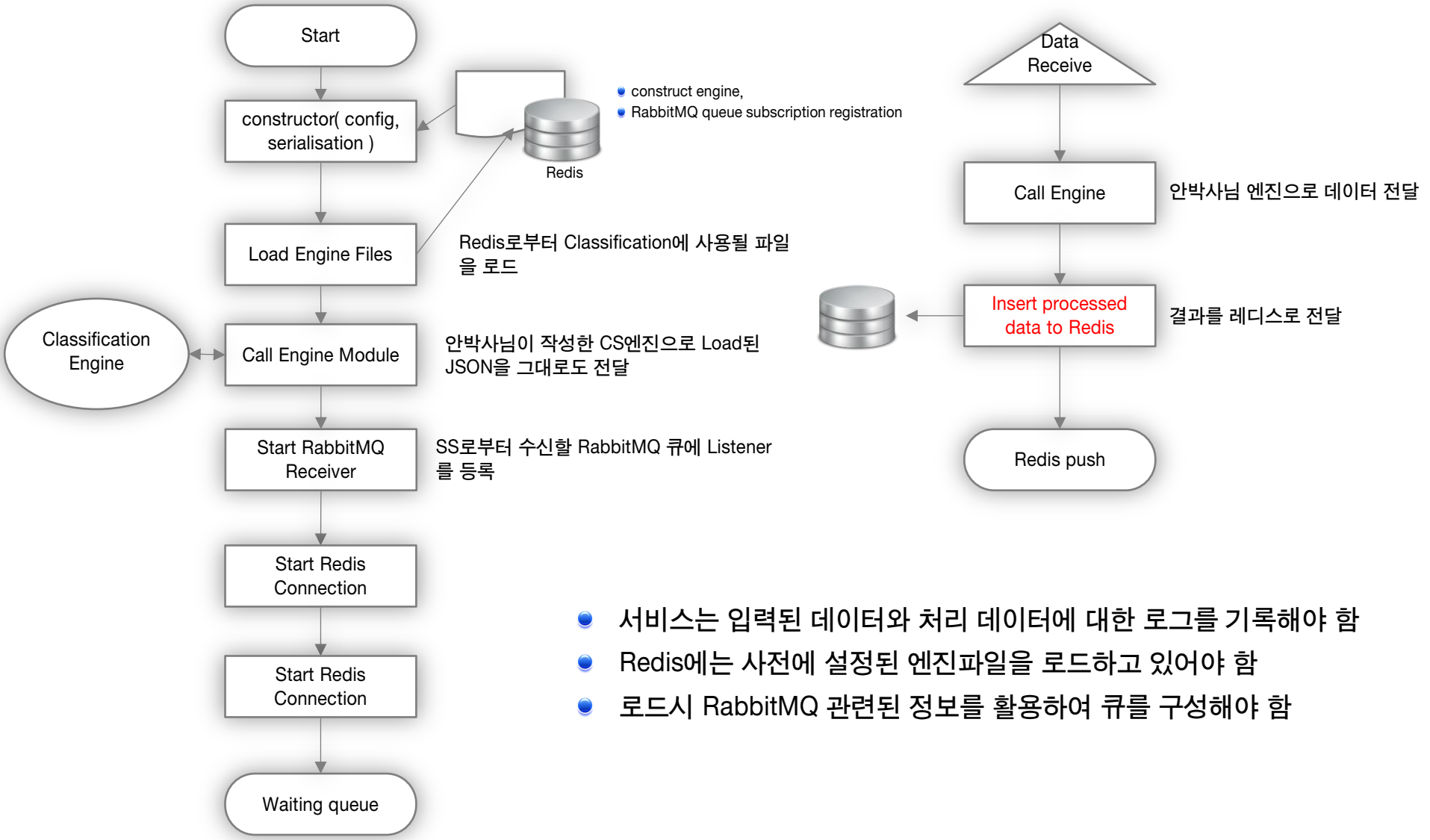
Data Service Flow



State Service Flow

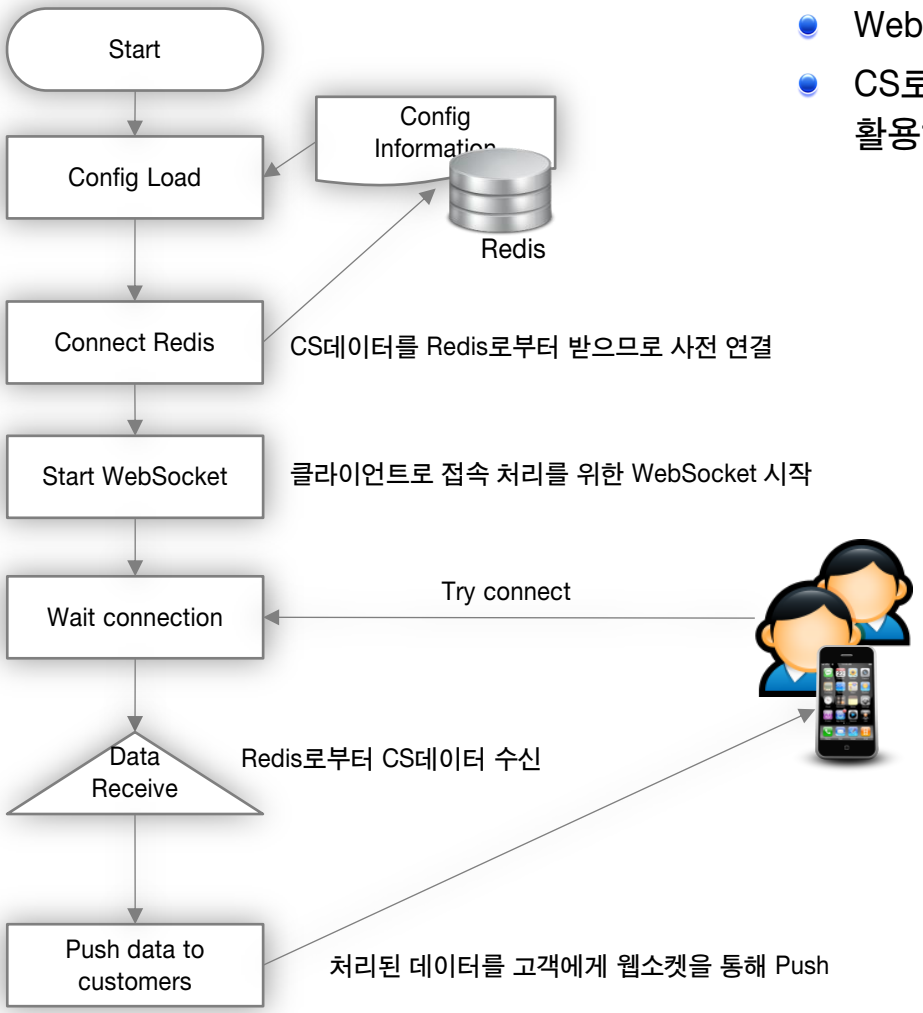


Classification Service Flow



- 서비스는 입력된 데이터와 처리 데이터에 대한 로그를 기록해야 함
- Redis에는 사전에 설정된 엔진파일을 로드하고 있어야 함
- 로드시 RabbitMQ 관련된 정보를 활용하여 큐를 구성해야 함

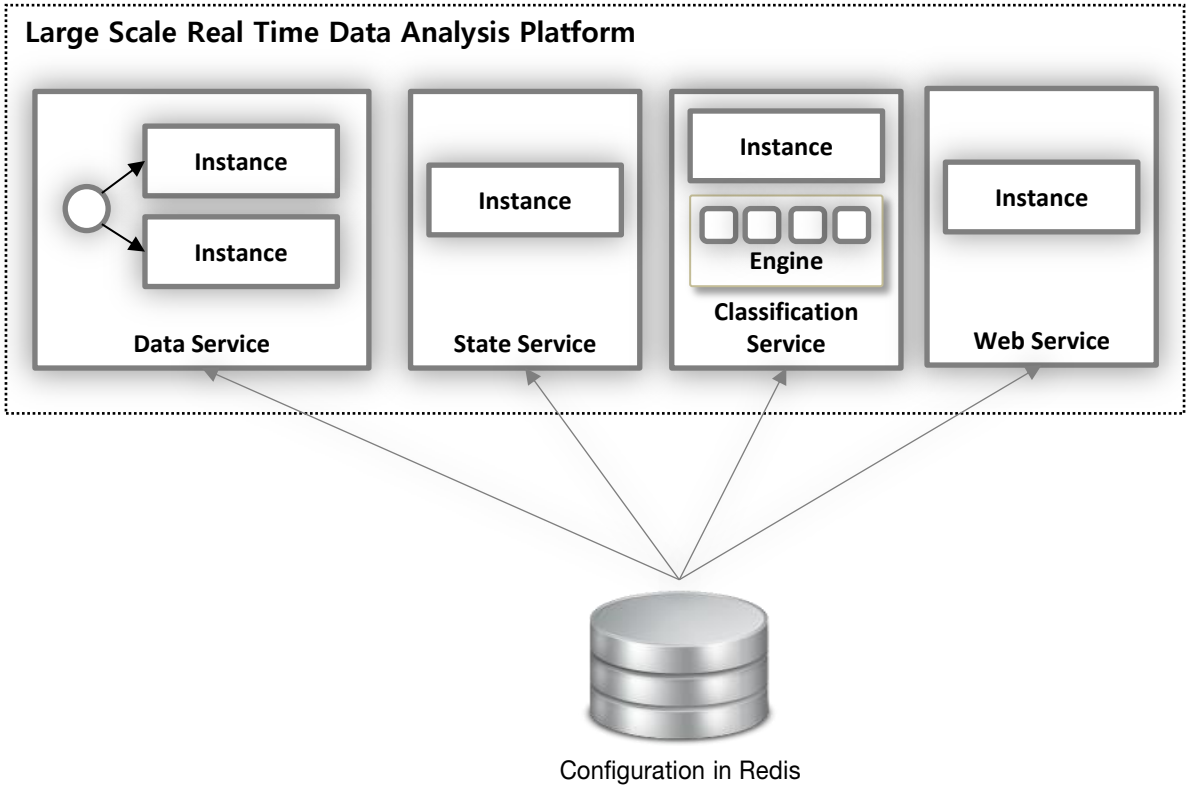
Web Service Flow



- WebSocket를 통해 클라이언트와 연결하고 있어야 함
- CS로부터의 데이터 수신은 Redis PUB/SUB 기능을 활용하여 처리하도록 함

Start Service Components

〈각 서비스 crontab에 등록〉



- python script를 활용하여 Redis로부터 필요한 설정정보를 받아서 각각의 Service instance 생성 및 run()을 호출
- python script 들로 executable 을 만들어 scheduler (crontab) 에 등록

Platform Requirement

Data Service

partition: d servers covering T / d topics each

process:

• to generate random numbers (seeded) to construct outbound me ssage per topic every P

seconds

• to publish the outbound messages when they are ready

State Service

partition: s servers covering T / s topics each

state model:

• list of data published from data service so far (ordered in time)

• averages of the data in different time periods

process:

• to subscribe messages under the topic belongs its coverage as well as relevant shared topics

• to update the state held in the instance

• to publish the outbound messages when the state is updated

• to update the state in cache

Classification Service

partition: c servers covering G / c engines each

classification model:

• maximum and minimum of the state

• constructor takes a dummy large text data

process:

• to subscribe messages under the relevant topics

• to classify the state

• to update the classification result in cache

Web Service

partition: as required

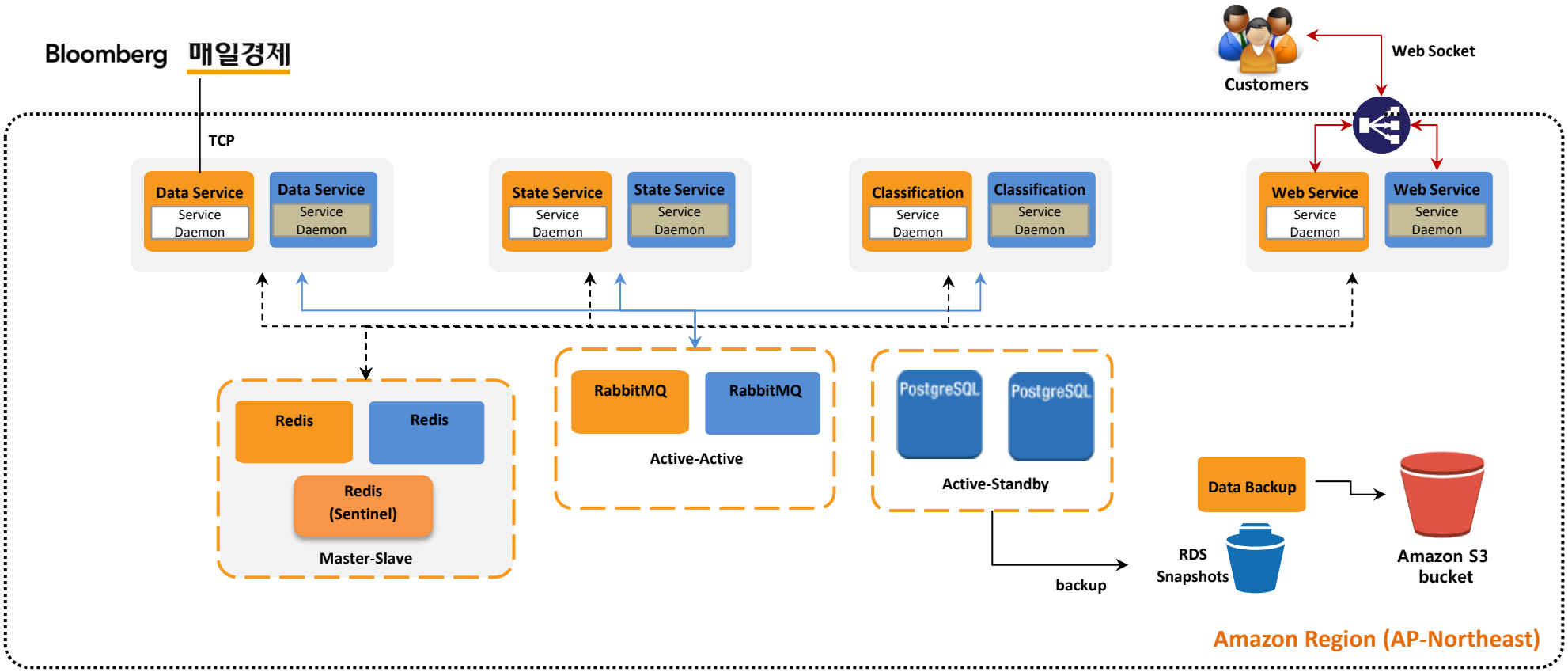
process:

• to cache client views

• to deliver the update if it is relevant to the client view

System Configuration
- Production on AWS -

Production Architecture on AWS



- 초기 m4.large(2 vCPU, 8GB Memory, EBS) 또는 m3.large(2 vCPU, 7.5GB Memory, SSD) 기반으로 시작
- 각 서비스에 대한 이중화 구성으로 데이터센터 장애에 대한 대비를 수행하도록 구성

Target Instances List

- 시스템에 대한 t2.large 서비스를 최소 기준으로 구성하여 초기 서비스 이후 상태 추이 추적 후 다운그레이드 적용

목적	서버 이름	인스턴스 유형	도메인 이름	지역	설명
Admin Server	prd-ap-admin	t2.medium	admin.buysignal.com	ap-northeast-1a	
Data Service	prd-ap-ds-1	t2.large/ m3.large	N/A	ap-northeast-1a	• 기본 2개의 서버로 초기 구성 • Elastic IP 할당
	prd-ap-ds-2	t2.large/ m3.large		ap-northeast-1b	
State Service	prd-ap-ss-1	t2.large/ m3.large	N/A	ap-northeast-1a	• 내부접속만 가능하도록 Security Group 구성
	prd-ap-ss-2	t2.large/ m3.large		ap-northeast-1b	
Classification Service	prd-ap-cs-1	t2.large/ m3.large	N/A	ap-northeast-1a	• 내부접속만 가능하도록 Security Group 구성
	prd-ap-cs-2	t2.large/ m3.large		ap-northeast-1b	
Web Service	prd-ap-ws-1	t2.large/ m3.large	api.buysignal.com	ap-northeast-1a	• Service에 따른 Auto Scaling 적용 • Web Socket을 위한ELB적용
	prd-ap-ws-2	t2.large/ m3.large		ap-northeast-1b	
RabbitMQ	prd-ap-rabbit-1	m3.large	N/A	ap-northeast-1a	• 기본 서버로 구성
	prd-ap-rabbit-2	m3.large		ap-northeast-1b	
Redis	prd-ap-redis-1	r3.large	N/A	ap-northeast-1a	• 2vCPU, 15G 메모리 구성 • Amazon Elastic Cache 검토 필요
	prd-ap-redis-2	r3.large		ap-northeast-1b	
Database	prd-ap-rds-master	db.m3.xlarge	N/A		• PostgreSQL RDS

Operating Cost Estimate on AWS

- 각 시스템간 이중화 구성 시 monthly 100% utilization 기준 2,640 USD 예상
- Weekly 40 hours 기준 서비스 구동 시 월간 비용 1,306 USD 비용 청구 예상

Services | **Estimate of your Monthly Bill ($ 2640.29)**

Estimate of Your Monthly Bill

Show First Month's Bill (include all one-time fees, if any)

Below you will see an estimate of your monthly bill. Expand each line item to see cost breakout of each service. To save this bill and input values, click on 'Save and Share' button. To remove the service from the estimate, jump back to the service and clear the specific service's form.

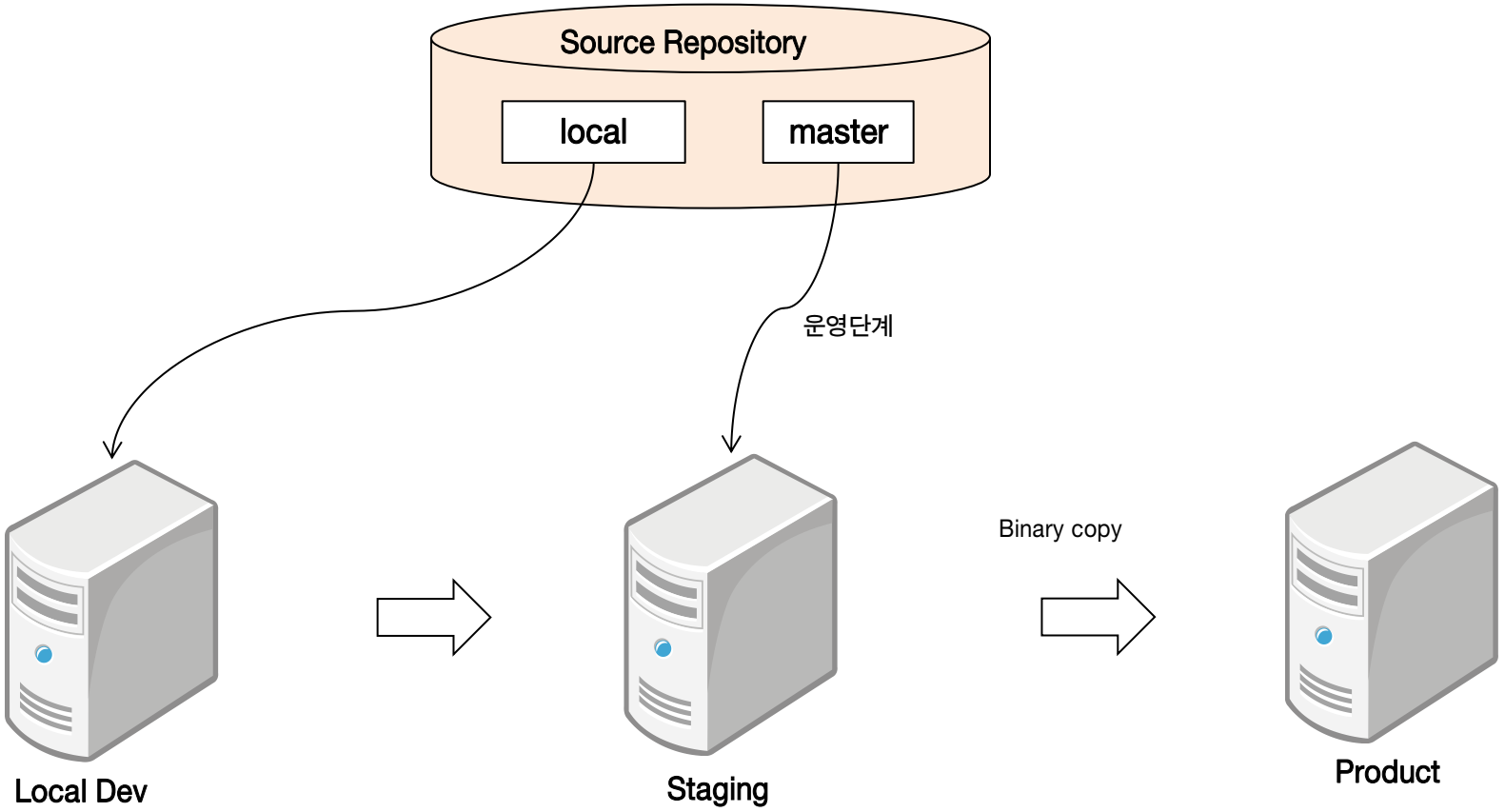
Save and Share

⊖	Amazon EC2 Service (Japan)		$ 1674.83
	Compute:	$ 1585.52	
	EBS Volumes:	$ 50.40	
	EBS IOPS:	$ 0.00	
	EBS Snapshots:	$ 0.76	
	Elastic IPs:	$ 18.30	
	Elastic LBs:	$ 19.77	
	Data Processed by Elastic LBs:	$ 0.08	
⊖	Amazon S3 Service (Japan)		$ 3.30
	Storage:	$ 3.30	
⊖	Amazon RDS Service (Japan)		$ 747.14
	DB instances:	$ 710.04	
	Storage:	$ 27.60	
	Backups:	$ 9.50	
⊖	Amazon CloudWatch Service (Japan)		$ 0.60
	Alarms:	$ 0.60	
⊕	AWS Data Transfer In		$ 0.00
⊕	AWS Data Transfer Out		$ 1.26
⊕	AWS Support (Business)		$ 240.03
Free Tier Discount:			$ -26.87
Total Monthly Payment:			$ 2640.29

Cost Estimate: http://calculator.s3.amazonaws.com/index.html#r=NRT&key=calc-633F3F2A-0C87-484B-8C1A-99AEEB2E5403

Deployment Plan

- Local Dev, AWS Dev Server 는 소스 저장소에 있는 최신 소스를 기반으로 Deploy를 수행 한다.
- Production server에는 staging에서 테스트 완료된 애플리케이션을 통한 복사를 통해 배포(장애 발생 요소 제거)



ø MQ Throughput – 2 vCore, 8GB Memory

	Sender	Receiver
1Kb Message	[root@localhost perf]# ./remote_thr.sh tcp://192.168.56.101:5555 1024 100000 Sent elapsed time : **2.339 sec**	[root@localhost perf]# ./local_thr.sh tcp://eth0:5555 1024 100000 message size: 1024 [B] message count: 100000 mean throughput: **39968[msg/s]** mean throughput: 327.417856[Mb/s]
2Kb Message	[root@localhost perf]# ./remote_thr.sh tcp://192.168.56.101:5555 2048 100000 Sent elapsed time : **1.499 sec**	[root@localhost perf]# ./local_thr.sh tcp://eth0:5555 2048 100000 message size: 2048 [B] message count: 100000 mean throughput: **49504[msg/s]** mean throughput: 811.073536[Mb/s]
4Kb Message	[root@localhost perf]# ./remote_thr.sh tcp://192.168.56.101:5555 4096 100000 Sent elapsed time : **2.137 sec**	[root@localhost perf]# ./local_thr.sh tcp://eth0:5555 4096 100000 message size: 4096 [B] mes sage count: 100000 mean throughput: **29446[msg/s]** mean throughput: 964.886528[Mb/s]
10Kb Message	[root@localhost perf]# ./remote_thr.sh tcp://192.168.56.101:5555 10240 100000 Sent elapsed time : **3.091 sec**	[root@localhost perf]# ./local_thr.sh tcp://eth0:5555 10240 100000 message size: 10240 [B] message count: 100000 mean throughput: **13048[msg/s]** mean throughput: 1068.89216[Mb/s]

Rabbit MQ Throughput – 2 vCore, 8GB Memory

- Producer/Consumer Simultaneously; 2K and 10K message

Producer/Consumer

```
[jboss@localhost rabbitmq-java-client-bin-3.5.5] ./runjava.sh com.rabbitmq.examples.MulticastMain --size
2048
starting consumer #0
starting producer #0
time: 6.000s, sent: 7198 msg/s, received: 7190 msg/s, min/avg/max latency: 1463/3982/23183 microseconds
time: 7.000s, sent: 7335 msg/s, received: 7341 msg/s, min/avg/max latency: 1508/2892/6231 microseconds
time: 8.000s, sent: 7386 msg/s, received: 7370 msg/s, min/avg/max latency: 1358/3691/15787 microseconds
time: 9.000s, sent: 7262 msg/s, received: 7270 msg/s, min/avg/max latency: 1306/3091/7463 microseconds
time: 10.000s, sent: 7283 msg/s, received: 7287 msg/s, min/avg/max latency: 1551/2924/7089 microseconds
time: 11.000s, sent: 7164 msg/s, received: 7170 msg/s, min/avg/max latency: 1565/3522/11979 microseconds
time: 12.000s, sent: 6043 msg/s, received: 6034 msg/s, min/avg/max latency: 1097/8846/41617 microseconds
time: 13.000s, sent: 6040 msg/s, received: 6051 msg/s, min/avg/max latency: 943/11451/58797 microseconds

[jboss@localhost rabbitmq-java-client-bin-3.5.5] ./runjava.sh com.rabbitmq.examples.MulticastMain --size
10240
starting consumer #0
starting producer #0
time: 5.000s, sent: 2447 msg/s, received: 2447 msg/s, min/avg/max latency: 550/1344/17682 microseconds
time: 6.000s, sent: 2701 msg/s, received: 2702 msg/s, min/avg/max latency: 569/1134/4756 microseconds
time: 7.000s, sent: 2732 msg/s, received: 2731 msg/s, min/avg/max latency: 534/1122/3344 microseconds
time: 8.000s, sent: 2728 msg/s, received: 2730 msg/s, min/avg/max latency: 551/1121/2082 microseconds
time: 9.000s, sent: 2798 msg/s, received: 2798 msg/s, min/avg/max latency: 606/1116/2105 microseconds
```

Solution	Description	
RabbitMQ	• Stable send/receive process • Can use AMQP(High performance) • Similar producer/consumer speed • Support runtime queue create/delete(very flexible)	

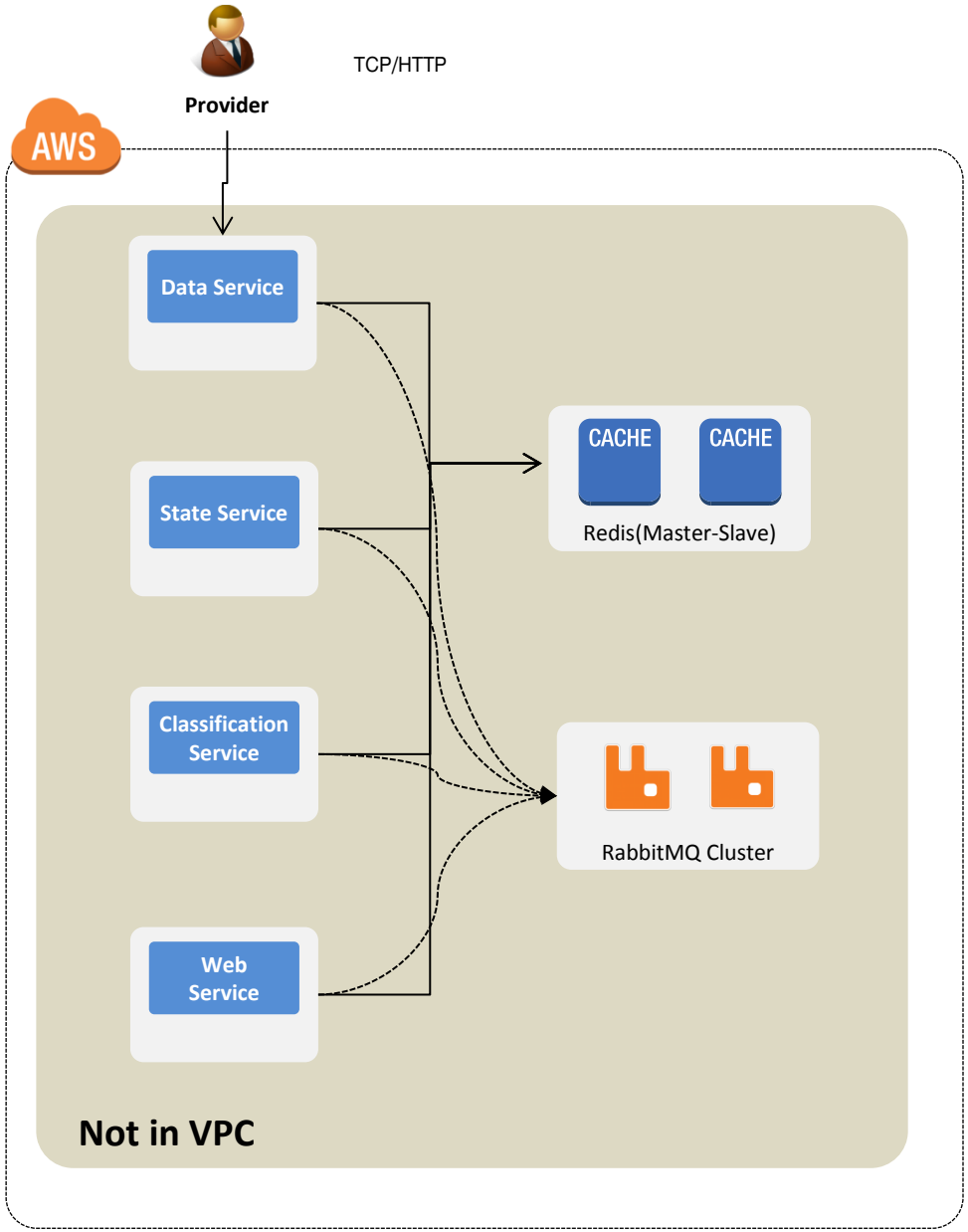
Comparison – RabbitMQ vs. ø MQ

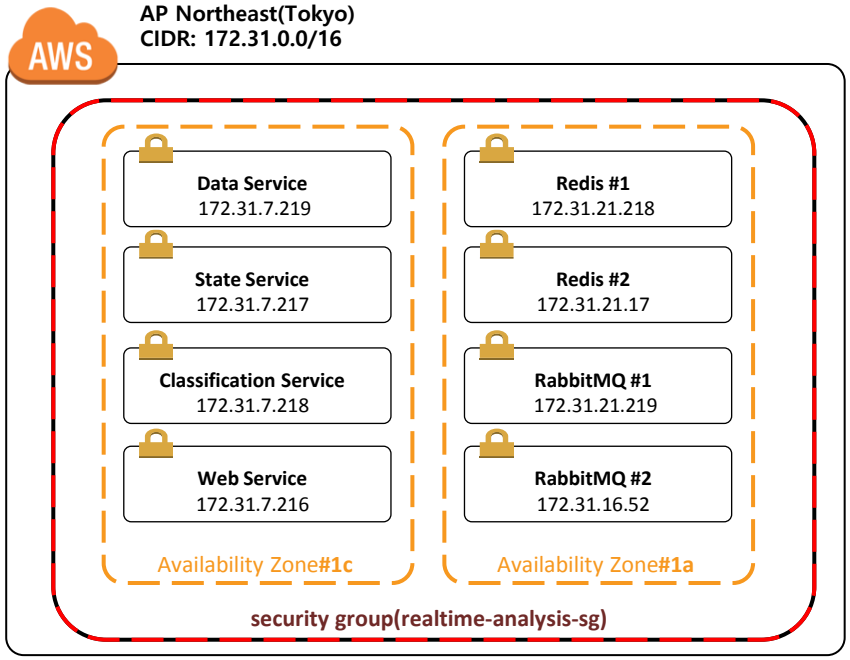
Criteria		RabbitMQ	ØMQ
Short Response Time	Short Messaging Routes	Short	Short
	Performance (Based on 10Kb)	2798 msg/sec	4.66 times faster than RabbitMQ – 13048 msg/sec
	Asynchronous Processing	Sync, Async Supported	Async based event processing
	Light-weight Software Stacks	Light	More Light
High Availability	No Single Point of Failures	HA Cluster Support	No Broker Architecture
Flexibility	Supported Data Formats	Various(JSON, POJO, etc.)	Various(JSON, Thrift, Google ProtoBuf, etc.)
	Easy Configuration	GUI Based Configuration	Implementation needed
	Runtime Configuration	Runtime deploy for queue, topic	Implementation needed
Commercial Support		SpringSource(VMWare)	iMatrix
Monitoring		Many plugin for monitoring, web based GUI	Implementation needed
Remarks		• Powerful web based monitoring • AMQP[1] Support • Dynamic topic management • Easy of Development	• High throughput/Low latency • Auto reconnect among peer • Run on arbitrary platforms (Windows, Android)

1) AMQP(Asynchronous Message Queuing Protocol) : Supported by Microsoft, Red Hat, VMware, Cisco, Novell, SoftwareAG, etc.

Failover Scenario

Amazon AWS Instances



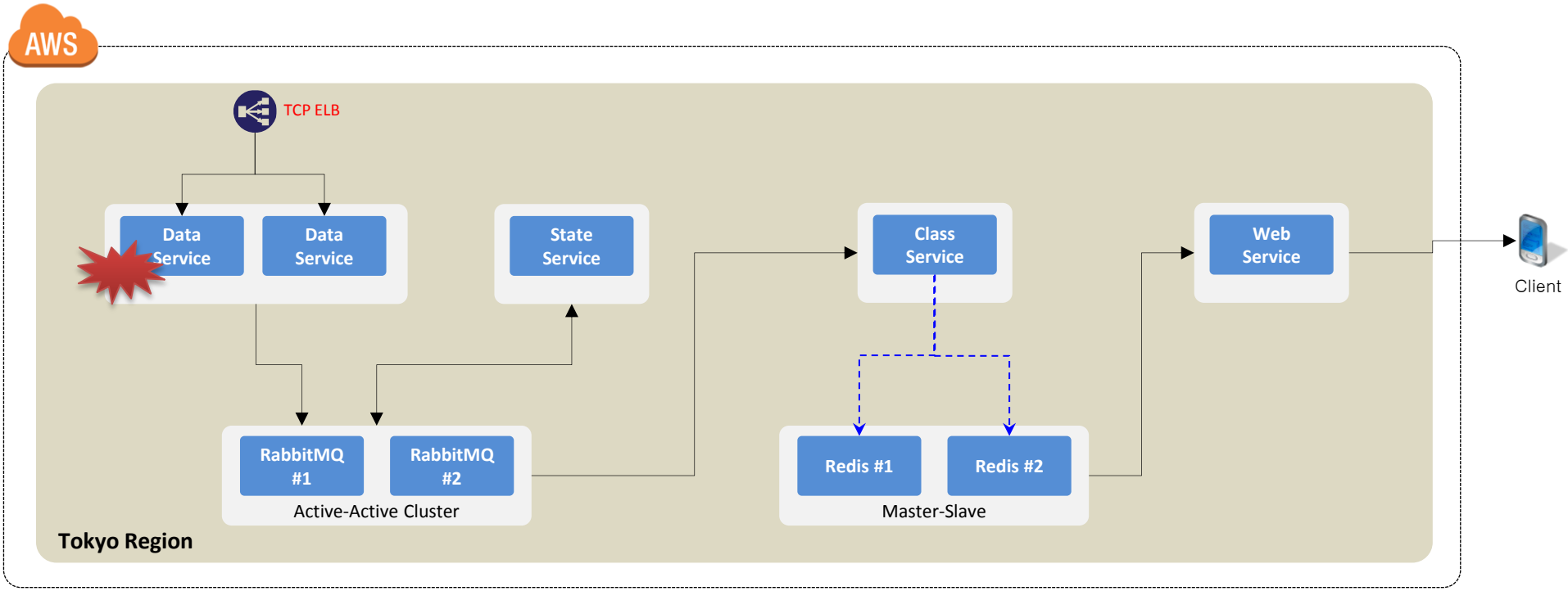


Name	Instance Type
Data Service	t2.medium
State Service	t2.medium
Classification Service	t2.medium
Web Service	t2.medium
RabbitMQ-1	t2.large
RabbitMQ-2	t2.large
Redis-1	t2.large
Redis-2	t2.large

사전 작업 고려 조건

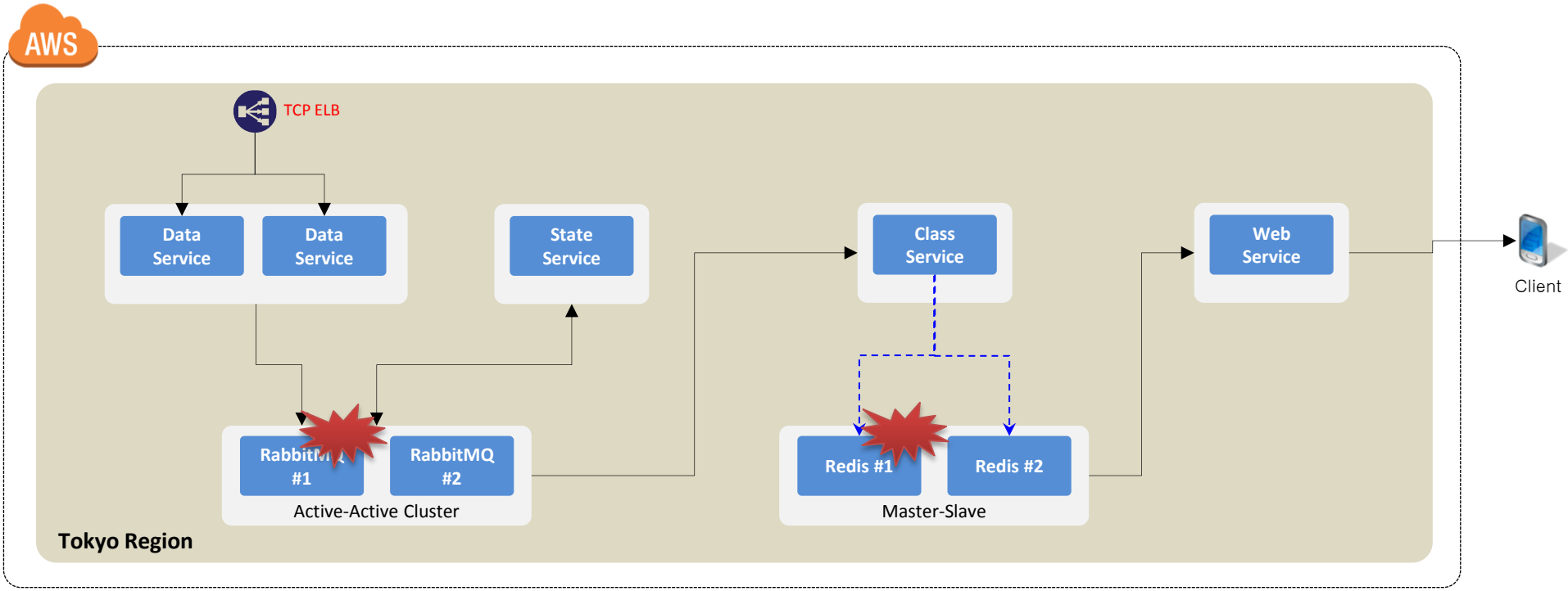
- 인스턴스
  - 클라우드 인스턴스 shutdown에 대비한 중앙 집중형 모니터링(Nagios, Zabbix 등) 솔루션을 통한 shutdown alarm 구성 및 인스턴스 재기동 프로세스
  - 인스턴스 shutdown에 대해서는 Cloud API 연계를 통한 template launch를 활용하여야 가능함
- Process
  - 프로세스들에 대한 상태 체크 모니터링 및 Process down시 재기동 스크립트를 각 인스턴스에 구성
  - 프로세스에 대한 체크는 crontab을 활용하여 처리
  - OS와 같은 다른 문제로 인하여 프로세스에 문제가 발생했을 경우 재시작을 한다고 해당 문제가 해결되지 않을 수도 있음

Scenario – Data Service Instance



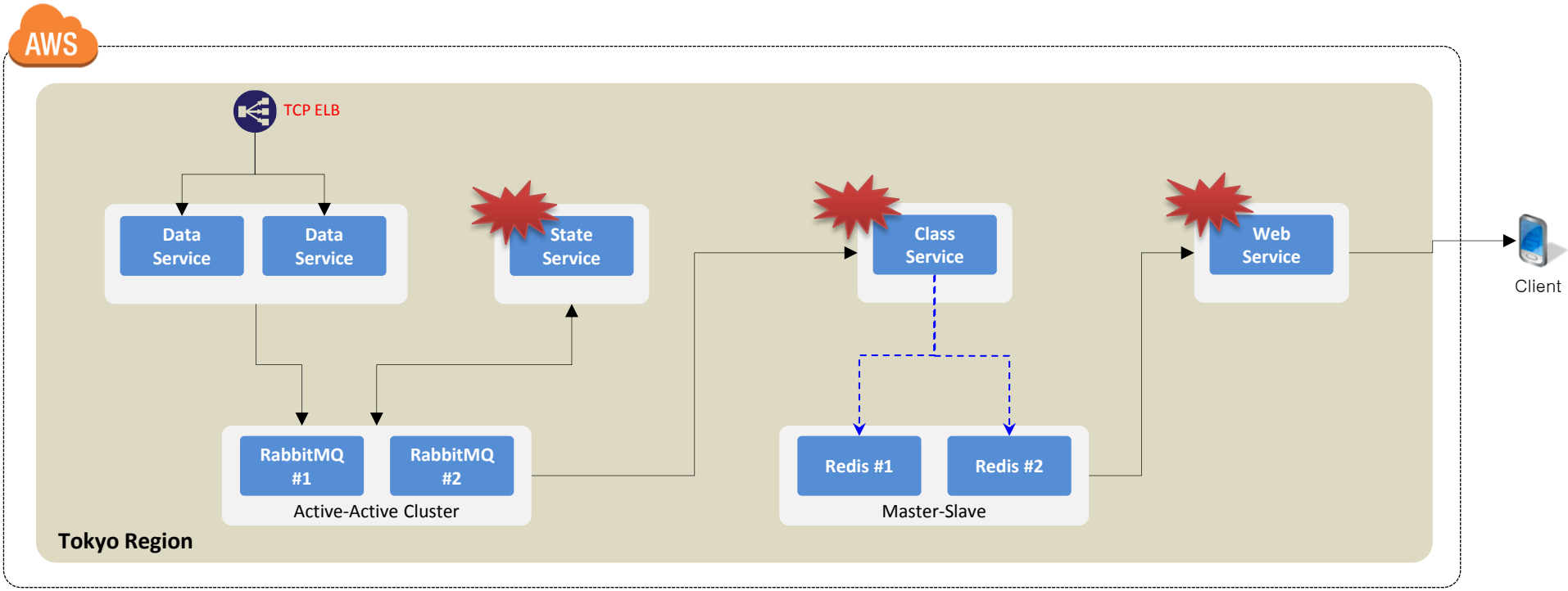
	장애내용	테스트 방법
1	Data Service Virtual Machine에 대한 인스턴스 장애	Data Service에 대한 EC2 Instance를 shutdown 시킴
2	Data Service Process에 대한 장애	Data Service Process에 대한 kill 명령 수행

Scenario – Middleware Instance



	장애내용	테스트 방법	예상 작동
3	RabbitMQ Active 서비스 장애	RabbitMQ Cluster의 main instance를 shutdown 시킴	RabbitMQ를 통한 지속적인 데이터 송수신 정상 기동
4	Redis Master-Slave 서비스 장애	Redis Master에 대한 EC2 Instance를 shutdown 시킴	Reids의 configuration 정보, 데이터 송수신 정상 기동
5	Redis Cache connection error	Redis 인스턴스에 대한 process를 kill시킴	서비스에서 정상적으로 slave 서버 접속 후 데이터 처리 확인

Scenario – State/Classification/Web Instance



	장애내용	테스트 방법	예상 작동
6	State 인스턴스 프로세스 장애	State Service의 process를 kill	crontab 데몬을 통한 감시 및 프로세스 재기동
7	Classification Service 장애	Classification Service의 process를 kill	Reids의 configuration 정보, 데이터 송수신 정상 기동

Source Code Directory Structure

Directory 구조 및 설명

/home/quant	설명	파일명	실행방법	비고
bin	C Code 변환 실행 파일	QuantManager	/home/quant/bin/QuantStart.sh	QuantStop.sh
	C Code 변환 실행 파일	QuantService	/home/quant/bin/QuantStart.sh	QuantStop.sh
	C Code 변환 실행 파일	WebSocketServer	/home/quant/bin/WebServerStart.sh	WebServerStop.sh
	C Code 변환 실행 파일	WebSocketPup	/home/quant/bin/WebServerStart.sh	WebServerStop.sh
etc	구동시 필요한 config항목 파일	quant.ini		
lib	공통 코드 구현체, confing class 구성	CommonClass.py		
	데몬으로 실행 시키는 class 파일	Daemon.py		
	QuantService에 상속되어지는 Class 파일	Rabbitmq.py		
	Rabbitmq에 상속되어지는 Class 파일	Redis.py	개별 파일로 실행 시 Python /home/quant/lib/Redis.py	- Redis에 환경 config 값을 등록 수정 - 차후 "set" function에 값을 수정 요망
	WebSocketServer 사용하는 참조 파일	wsbridge.py		
log	차후 로그 수집 디렉토리			
cbc(callback)	예전 callback 디렉토리를 cbc로 변경	DataCallBack.py StateCallBack.py ClassCallBask.py	QuantServer(ServiceSlass.py) 상속됨	Service Type에 맞춰 파일의 내용을 수정합니다.
web	Css, js, index.html만 남기고 모두 삭제		quant.js에서 WebSocket 구현	
src				
-/component	Service 구동 원천 소스	QuantManager.py ServiceClass.py WebSocketPub.py WebSocketServer.py	- QuantManager - QuantService(SeviceClass.py) - WebSocketPub - WebSocketServer	Cython로 embed된 파일 만들어집니다.

Source Code Directory Structure

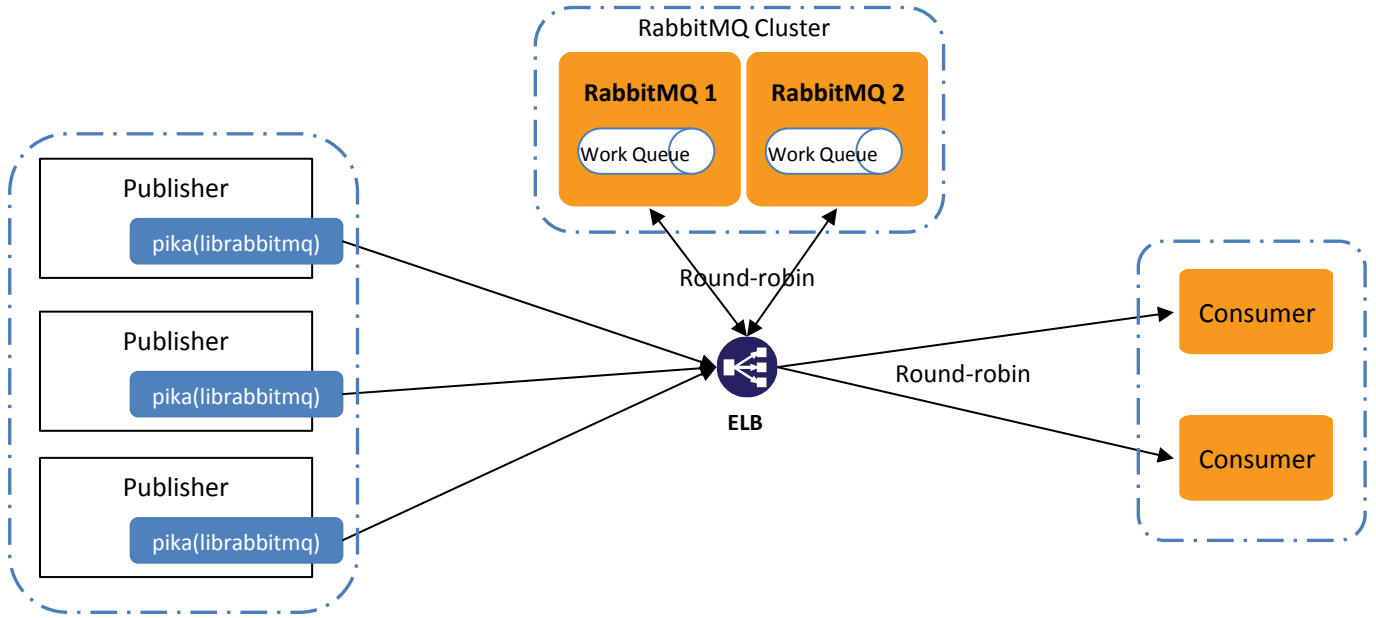
- Directory 구조 및 설명

/home/quant	설명	파일명	실행방법	비고
-/make	Cython embed 파일 생성 script	make.sh	./make.sh or sh make.sh	
-/install	rebbitmq 설치 script입니다.	rabbitmq-install.sh	sh rabbitmq-install.sh	Erlang, rabbitmq의 yum 설치합니다.
	pip로 python 관련 모듈을 설치합니다.	pip-setup.sh	sh pip-setup.sh	Zeromq, redis 등 필요한 library를 설치합니다.

Appendix
- Platform TO-BE Architecture Guide -

RabbitMQ High Availability

- Make RabbitMQ Instances as a cluster

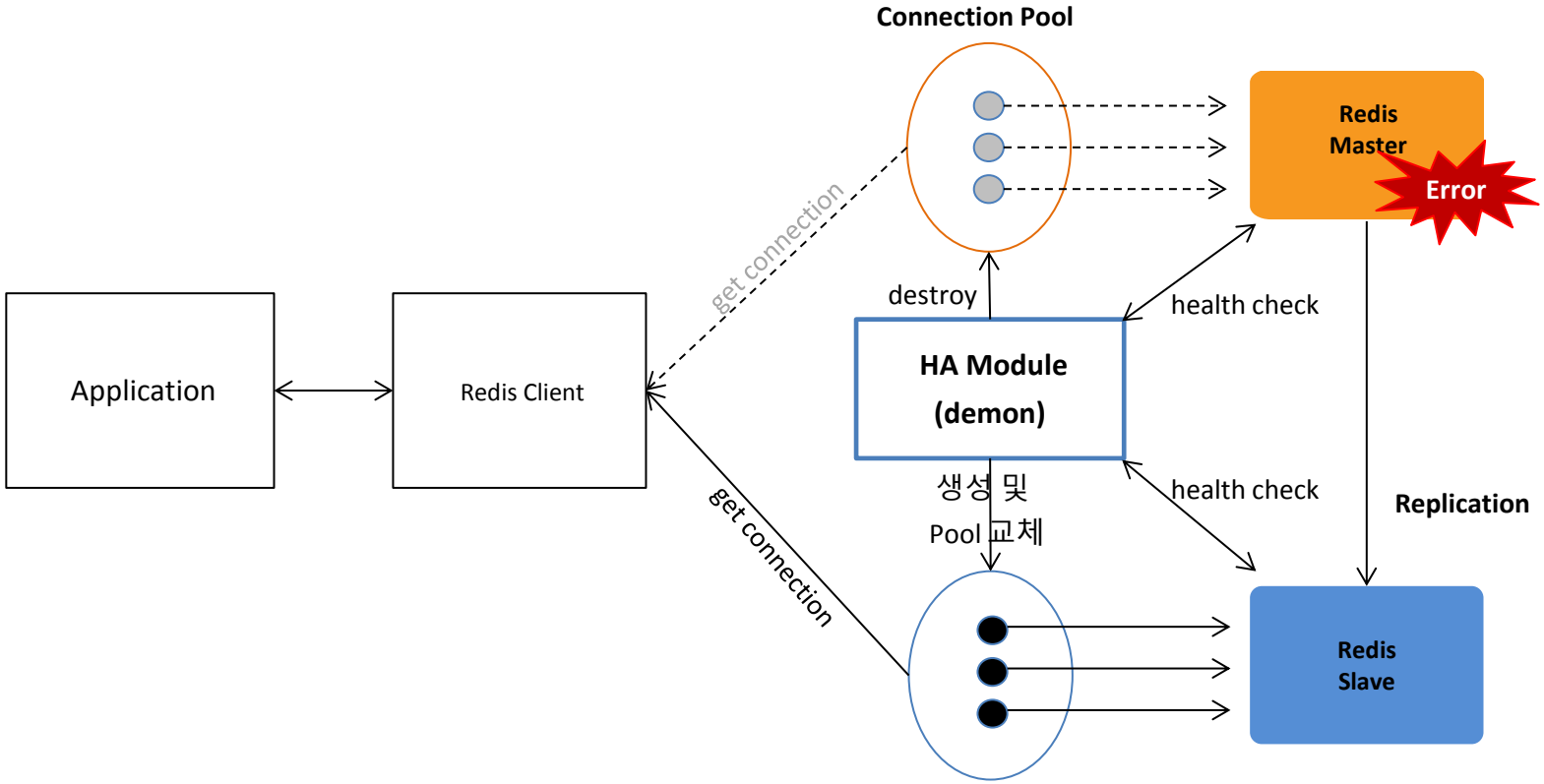


- Features

✓ 각 Publisher 및 Consumer 에서는 각각 하나의 connection 을 재사용
✓ Scalability : RabbitMQ Cluster 안의 노드 수를 동적으로 구성할 수 있으며, 각 서비스 또한 동적으로 확장 가능
✓ Message Durability : RabbitMQ Message Queue 를 파일에 저장하거나 노드들간 복제를 통한 메시지 유실방지
✓ 범용적인 메시징 시스템으로 활용 가능

Redis High Availability with Sentinel

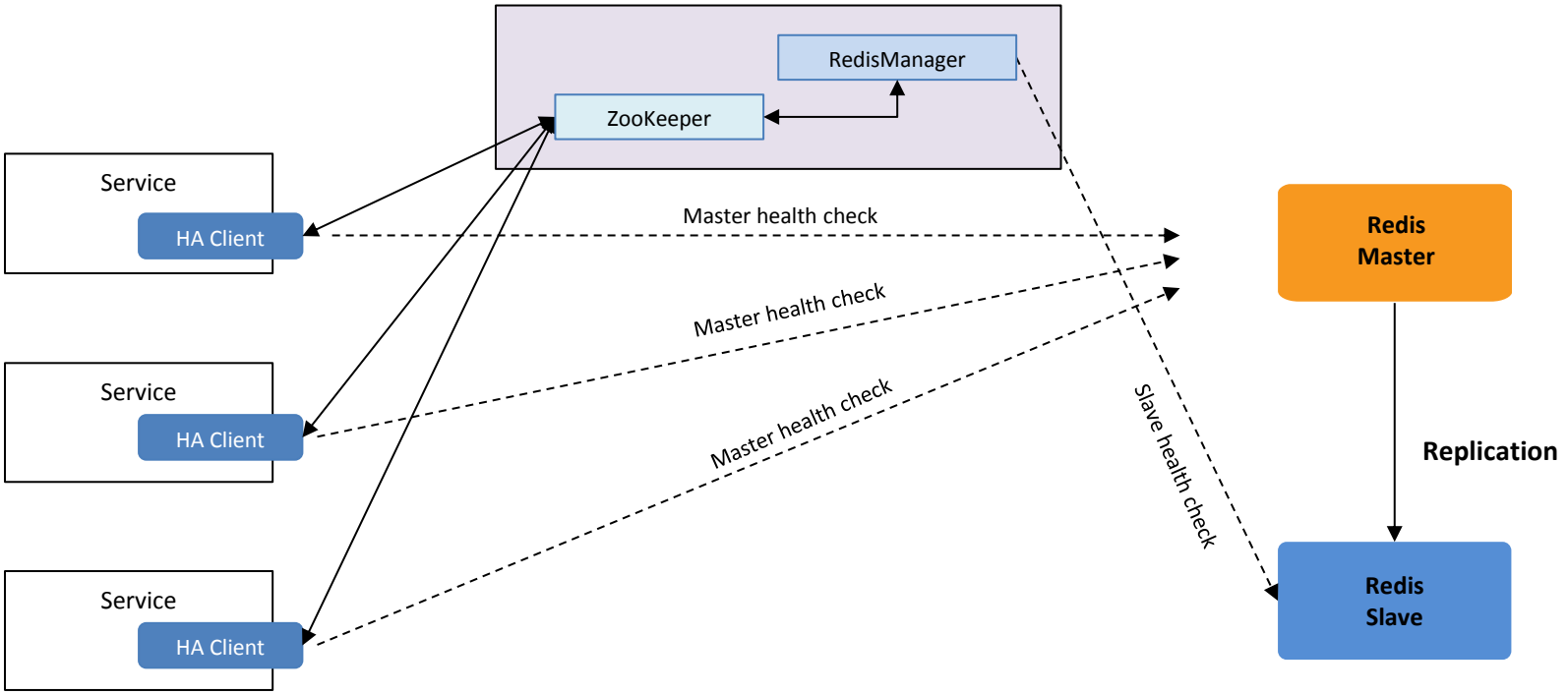
- Configure Redis master/slave using sentinel to make high availability



✓ 주기적으로 master 서버와 slave 서버를 모니터링(health check) 하고
✓ 예기치 않은 장애 발생으로 master서버에 문제가 발생하는 경우 sentinel은 slave를 신규 master 서버로 승격시킴
✓ 기존 master 서버 connection pool을 신규 connection pool로 교체하고
✓ 기존 connection pool의 자원을 해제한다. (내부적으로 기존 master 서버를 slave 서버로 관리함)
✓ 기존 master 서버 장애 복구 후 slave 서버로 재기동 (수동작업)

Redis High Availability with ZooKeeper

Architecture

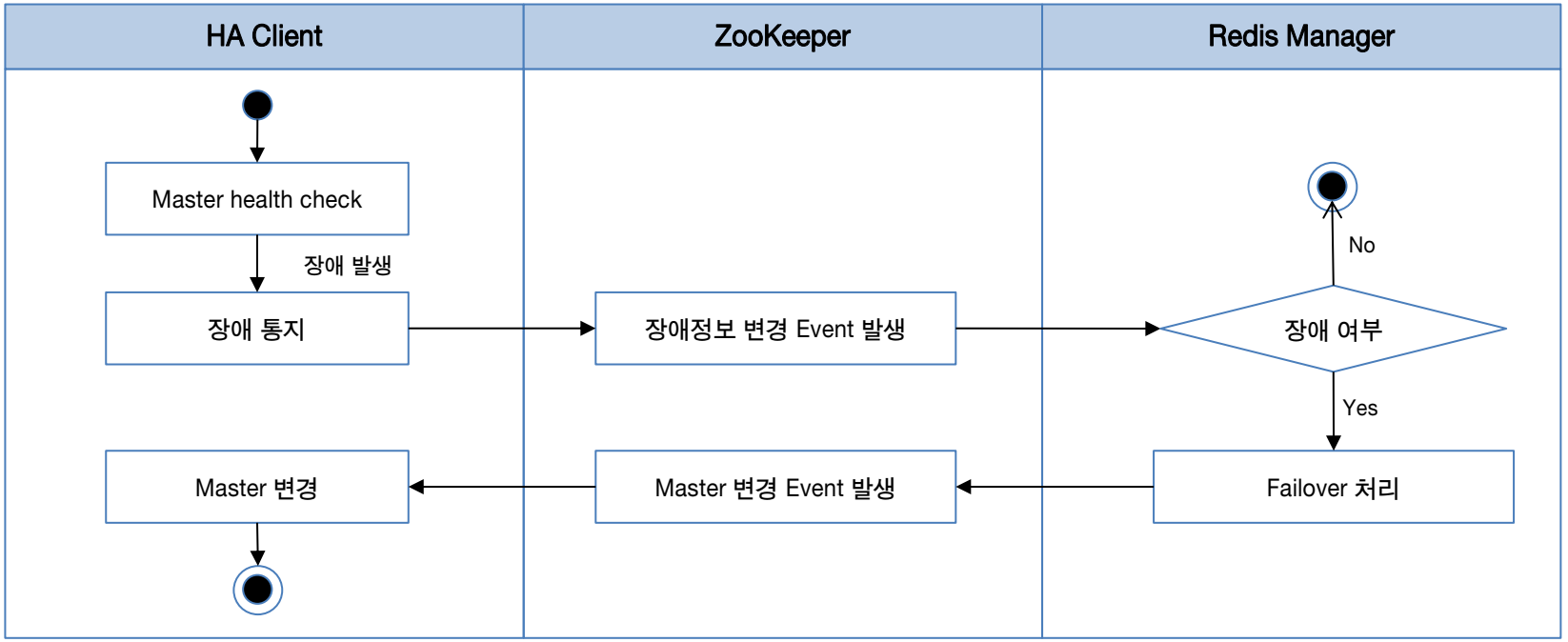


Features

- ✓ Redis 접속설정은 ZoooKeeper Server 에서 통합관리
- ✓ Redis Client Server 중심의 장애 상황에 따른 Auto Failover를 수행. (by RedisManager)
- ✓ ZooKeeper Server 장애 시에도 정상적인 서비스가 가능

Redis High Availability with ZooKeeper

- Make Redis HA without sentinel



✓ HA Client 의 장애 통지는 ZooKeeper 의 /db/redis/master/[*client znode*] 의 data 를 변경하는 작업.

✓ Redis Manager 의 Failover 처리

✓ HA Client 의 Master 변경은 새로운 Master 주소로의 connection pool 을 재생성함.

OPEN

SHARE

CONTRIBUTE

ADOPT

REUSE