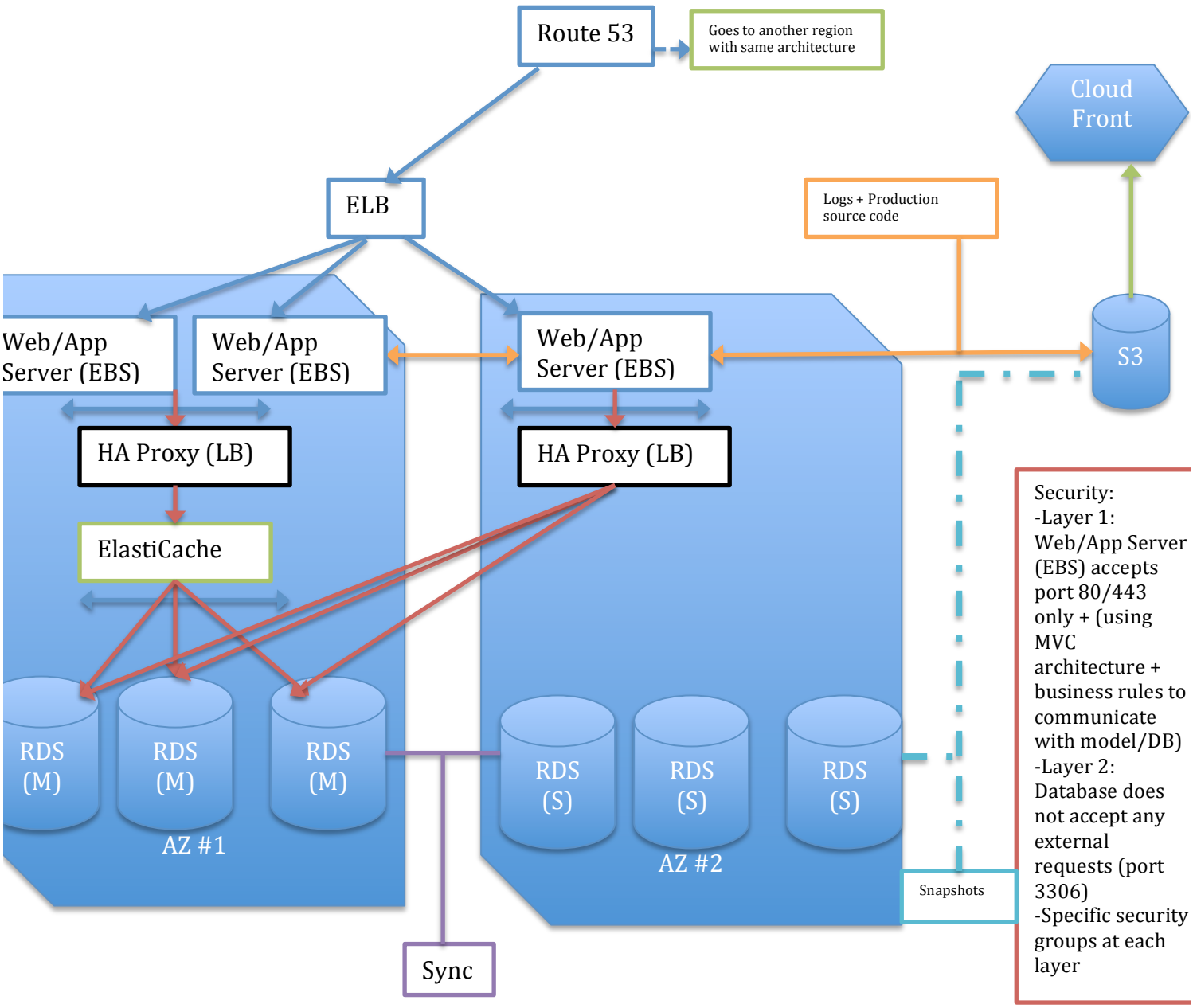
<u>An approach to SaaS multi-tenant app architecture in AWS</u> *(Version 1)*
Customers choose their region-specific, allowing the data to abide to the legal terms per region.



SNS for notifications about EC2, RDS, ElastiCache, spot instances for cron,...

SQS to help in queening threads/requests between different controllers

SWF to automate signup process that works and decides upon communicating with control panel, CRM, Ticketing/Billing Svstem

SES to handle all email requests to be sent from tenants

simpleDB to help EC2 launch configuration and assist in monitoring their status

CloudWatch to alert on heavy/low utilization of EC2, RDS alarms, monitoring,...

CloudFormation is used to replicate this architecture into another region

eliematta.com

General information:

-The first split is handled on the **R53** DNS (eg: companyname.website.com)
-**ELB** detects high latency and redirects to healthy instances while also being resilient for fault tolerance.
-Usage of **Elastic IPs** in case of replacing a Web/App server via API
-The second split of tenant in this region is handled on the App layer (which will help depicting the rightful data to show from ElastiCache and RDS)
-Web/App servers are **EBS-backed** for launching instances faster and for high I/O EBS-Optimized (between EC2 and EBS), they can be stopped and the EBS can be used as HDD. In other cases, negligible cost savings while using instance-store backed AMI is not preferred.
-**S3** volume saves the data in one region, contains production source code, css, images, js, logs, snapshots,...
-**CloudFront** distributes data geographically (multiple regions) from an Origin Server, which is in this case, S3.
-**Auto Scaling** is set to scale out when usage reaches 80%, scales back when usage reaches 10%
-**ElastiCache** uses **Auto Discovery** that will help the App to know the rightful owner of the data vis-à-vis of the requester using DNS/CNAME for each of the cache node endpoints.
-Data is persisted in **RDS** Master cluster (tenant specific RDS) and synced with Slave for HA to another AZ.
-Fault tolerant system is taken into consideration as there is no single point of failure, alarms/notifications are set and components are loosely coupled via SQS and MVC App component-based architecture. (PS: Failover take action via ELB to another EC2 instance or AZ, DB failover to another AZ, Auto Scale takes care of redundancy (min-max). More work can be done, discussed in *Future Work*)

EC2 launch:

-**Bootstrapped** using initial layers included in AMI + applies chef recipes concerning app configuration.
-Instance info is taken from **simpleDB** using domain queries when passing secure AWS key/encrypted credentials and instance-specific parameters for the EC2 instance, this also updates the instance status for the retrieved row.
-EC2 applies chef recipes and access S3 to create/get the latest .zip package.
-EC2 AMI is saved via incremental snapshots.

Cron jobs:

1-**Scanrio 1**: Updating tenants
-Run update tenant job to populate a certain change in database via custom script, which is a db-transparent migration command.
-This leverages the usage of SQS with spot instances bidding a bit more on on-demand prices. Wait 5 mins before shifting to on-demand.
-Interruptions/Fault tolerance is taken into consideration since it is transaction based changes and allows the usage of Queue/still alive check.
-simpleDB helps in passing tenant-specific configuration.

2-**Scenario 2:** Updating data-specific per tenant

-Run daily cron jobs that will execute a job to update custom recommendations for each tenant on a specific time per day.
-This leverages the usage of spot instances that runs a recommendation rating algorithm based on Bayesian rating to update recommendations for each tenant.
–Interruptions/Fault tolerance is taken into consideration using Grid or Queue architecture.
-simpleDB helps in passing tenant-specific configuration.

Deployment and Testing

-Setup the previous architecture to be formed in OpsWorks. (Pending, needs more research)
-Deployment on test environment via commit to OpsWorks. (Pending, needs more research)
-Deployment to production on all instances.

In case OpsWorks was not suitable, check Elastic BeansTalk or Do It Yourself as follows:
-Have an EC2 on-demand/reserved small instance for testing that will have the latest source code deployed from git/svn.
-In case of going live, the script generates a .zip package and deploys it in S3.
-The script lists all Auto-Scaling groups having running instances in all regions. Search, stop, terminate and launch new instances (that will automatically take the latest .zip package from S3 via bootstrapping) + update Elastic IPs
-QA is done using spot instances/on-demand.
-Testing/Continuous delivery via Jenkins and spot instances.

Cost:

~300$/month = 1 client, 1 region
~2000$/month = 50 clients, 1 region

Future work and amelioration:

-Consider using **Elastic BeansTalk** for a more flexible on-demand production/development environment that manages versioning. Deploys automatically on commit and manages internally: Auto Scaling, ELB and EC2 instances.
-Consider using **OpsWorks** that can make the maintenance easier of the whole process while giving direct access to chef recipes for more bootstrapping capabilities.
-Consider using **Data Pipeline** when it is available in all regions to be used in case of DR backup from one region to another. (compared with corn jobs with SQS or SWF)
-Consider using **VPC with public and private subnets** for keeping the application and database servers not accessible directly from the Internet, while still being able to be accessed via NAT for patches, ...
-Consider dividing tiers physically between Web server and App server for more security/scalability/loosely-coupled concerns.
-Consider **Disaster Recovery** and leverage the fact of copying EBS between regions, this requires setting up a special script that dumps data and save it to

another volume in another region. Also transferring clients to another region in case of DR and creating DBs based on those snapshots.
-Using MapReduce/Hbase for recommendations system for a faster result.