intetics

Where software concepts come alive™

Examples of Architecture Design and Technology Selection

White Paper

Solutions

Intetics is a leading global technology company focused on the creation and operation of effective distributed technology teams aimed at turnkey software product development, digital transformation, quality assurance, and data processing. Based on a proprietary business model of Remote In-Sourcing®, proprietary Predictive Software Engineering framework, advanced Quality Management Platform, measurable SLAs, and unparalleled methodology for talent recruitment and retention, Intetics enables IT rich, innovative organizations to capitalize on available global talent and Intetics' in-depth engineering expertise. At Intetics, our outcomes do not just meet clients' expectations, they have been exceeding them for our 27+ years in business. Intetics is ISO 9001 (quality) and ISO 27001 (security) certified. The company's innovation and growth achievements are reflected in winning prestigious Inc 5000, Software 500, CRN 100, Deloitte Technology Fast 50, GSA, Stevie People's Choice, Clutch, ACQ5 and European IT Excellence awards, and inclusion into IAOP Best Global Outsourcing 100 list. You can find more information at https://intetics.com

Automate the EC2 Image Update Process

Description

- Reduce the manual work of updating EC2 instances; use new custom AMI images to better comply with requirements from the security team.

Problem

- The project team received requirements from the Client that EC2 instances should use new AMI images right after the new AMI image was built.
- Previously, it was manual work that required updating CloudFormation code and re-creating instances. New AMI images could appear at any time, and the team was interrupted from their current work to update EC2 instances.

Solution

- The Client already used the EC2 Image Builder service to build new AMI images.
- The team proposed a serverless solution. It is based on SNS, Lambda, Step Function, DynamoDB, and SES AWS services.
- EC2 Image Builder publishes events to the SNS topic when a new AMI image is ready.
- Step Function is subscribed to the SNS topic, and when a new event appears it:
    1. Calls Lambda to update the AMI image for all Autoscaling groups.
    2. Scans for all existing EC2 instances and writes information to DynamoDB. Also, it updates all Autoscaling groups and increases the desired instance number.
    3. Sleeps for 3 minutes.
    4. Terminates old EC2 instances.
    5. Sends an email with a report about the instance update status.
- To make sure that instances were terminated, DynamoDB Streams were used.

Outcome

- New AMI images are applied to all EC2 instances automatically and without delay.
- The team is no longer interrupted from business features by the AMI image update work.

TCO Calculation

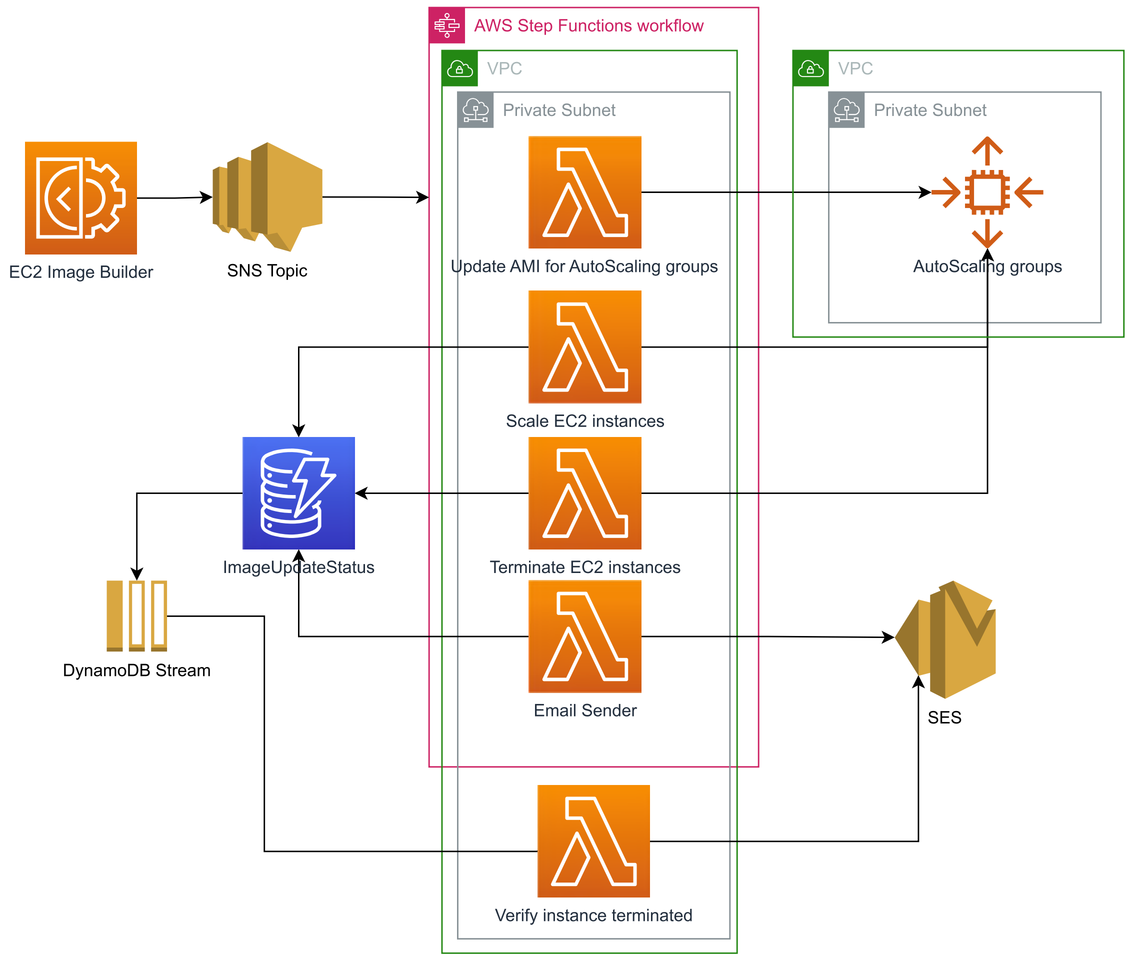
- The current solution uses fully managed AWS services, and the expected cost for a solution was calculated as less than $50 per month, assuming that AMI images will be updated once per month. This cost is based on EC2 instance recreation, Lambda invocation, and DynamoDB.

  This information was discussed with the Client before implementation.

Lesson Learned

- Fully managed services can significantly reduce maintenance work.

Solution Details



- The Client already used the EC2 Image Builder service to build new AMI images.
- The project team received requirements from the Client that EC2 instances should use new AMI images right after the new AMI image was built.
- The team proposed a serverless solution that was based on SNS, Lambda, Step Function, DynamoDB, and SES AWS services.
- EC2 Image Builder publishes an event to the SNS topic when a new AMI image is ready.

Step Function was subscribed to the SNS topic, and when a new event appears, it:

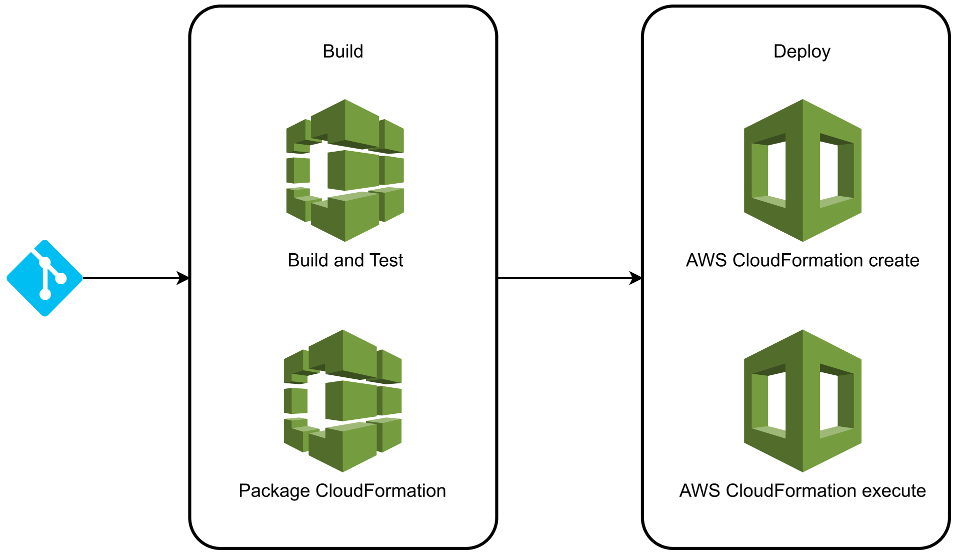
1. Calls the "**Update AMI for Auto Scaling groups**" Lambda. This Lambda updates the AMI image for all Auto Scaling groups (by creating a new launch configuration).
   Next, it calls the "**Scale EC2 instances**" Lambda. This Lambda scans for all existing EC2 instances and Auto Scaling groups and writes information to DynamoDB. This Lambda also updates all Auto Scaling groups and increases the desired instance number to x2 from the current value.
2. Step Function invokes the "Sleep" step to wait for 3 minutes until new instances are created and initialised.
3. After the timeout, Step Function invokes the "**Terminate EC2 instances**" Lambda. This Lambda terminates old EC2 instances.
4. As the final step, Step Function calls the "**Email Sender**" Lambda, which sends an email with a report about the instance update status.
5. Additionally, the solution has a "**Verify instance terminated**" Lambda, which listens for the DynamoDB stream and checks that EC2 instances for expired and removed records are terminated. If not – then it sends an email to manually terminate the EC2 instance.

AWS Services That Were Used for the Solution

- The solution contains 5 Lambdas, and each Lambda has its own purpose.
- DynamoDB is used to store historical information about the instances and Auto Scaling groups that were updated.
- SNS is used to initiate processing.
- Step Function is used to orchestrate Lambdas.

Deployment Process

- Git is used as the version control system to store source code (including CloudFormation code and buildspecs for CodeBuild) for this solution.
- CloudFormation is used to describe AWS resources. The AWS Serverless Application Model (AWS SAM) is used as a CloudFormation extension to get reliable deployment capabilities. AWS Cloudformation templates are split into logical stacks, so modules are decoupled, reusable, and easier to maintain. Templates also take advantage of nesting and/or cross-stack references.
- The CloudFormation validate-template command is used to validate the template at the build stage of CodePipeline.
- CodeBuild is used to build applications, run unit and integration tests, and do static code analysis.
- CodePipeline is used for Continuous Deployment.



Roll Back Process

- Versioning for Lambdas is used to keep the previous version. It allows for rolling back to a particular Lambda if there are issues after release.
- The current solution doesn't have Automation tests, as it performs operational work and doesn't have any business logic/UI/API. Functionality from Lambda functions is covered with unit and integration tests.

Application Monitoring

- CloudWatch Dashboard is used to show the overall status of the solution. Default CloudWatch metrics (for Lambda, DynamoDB, SNS, Step Function, and CodePipeline services) cover all requirements for metrics for the current solution.
- CloudWatch Dashboard is used to visualise how the solution works.
- CloudWatch alarms are configured to send degradation and outage events in case of errors.
- CloudWatch Alarms are configured on Lambdas – Error count and Duration; Step Functions – failed executions.
- CloudWatch logs and X-Ray are used to investigate issues.
- All logs from Lambdas are available in CloudWatch logs. We use the INFO log level.
- X-Ray is configured for all Lambdas.
- Reserved concurrency was configured on all Lambdas to 1, as actions are performed sequentially.

Security

- Pylint is used for static code analysis. It's performed as part of the Build step in AWS CodeBuild.
- The credentials for the root user contain only one person from the Client team. IAM Users are created for each team member from Intetics who need access to the AWS account. Access keys are not assigned to the root user. MFA is enabled for the root user, as well as for IAM Users.
- The current solution does not require Internet access. All Lambdas are added to private subnets in VPC.
- AWS services were used as IAM principals: application-autoscaling.amazonaws.com, lambda.amazonaws.com, and sns.amazonaws.com.

Security Groups

- For this solution, we have three types of security groups: for CodeBuild, individual security groups for each VPC Lambda, and security groups for EC2 instances.
- Security groups for Lambdas and CodeBuild have:
  - no inbound rules
  - allows all traffic in outbound rules
- For EC2 instances in Auto Scaling Groups, we have individual security groups per ASG with:
  - inbound rules: allows http access (80 port) for ALB (using ALB Security Group Id for Source)
  - outbound rules: allows all traffic

Auditing

- CloudTrail is enabled in all AWS Regions.
- CloudTrail log file integrity validation is enabled.
- CloudTrail log files are stored on S3 on a separate AWS account.
- Versioning is configured on S3 with logs.

DynamoDB

- The main requirements were to use a fully-managed service that does not require any maintenance work, has automated backups, can scale on-demand with a pay-as-you-go payment model, and can automatically remove old records and publish events on deletion. DynamoDB matches all these requirements.
- Single DynamoDB is used to store all data for the solution.

- It is used to store:
  - AutoScaling group information
  - EC2 instances information, including TTL
  - AMI Image information
- We use TTL for EC2 instance items in DynamoDB to clean up old EC2 information. To make sure that these instances with old AMI images were terminated, we use DynamoDB streams. Lambda subscribes to DynamoDB streams and verifies that physical EC2 instances were terminated for deleted EC2 instance items from DynamoDB.

DynamoDB Structure

- To simplify the development process for the solution, data for DynamoDB was designed as Relational Data (like in this example).
- DynamoDB structure:
  - Primary partition key with String type and HASH key type
  - Two GlobalSecondaryIndexes GSI-1 and GSI-2 for Attribute names Data1 and Data2 and projection type – ALL
- Such a structure allows for storing different entities. Global Secondary Indexes are used to improve data access, and they allow for using Query for all read patterns and to avoid scan operations.
- Main Entities:
  - AutoScalingGroup
    - PK - ASG~<AutoScaling Group arn>
    - Data1 - APP_NAME~<application name>
    - Data2 - AMI~<AMI arn>
  - EC2Instance
    - PK - EC@_~<EC2 instance id>
    - Data1 - SOURCE_ASG~<AutoScaling Group arn>
    - Data2 - AMI~<AMI arn>
  - AMIImage
    - PK - AMI~<AMI arn>
- Other DynamoDB attributes are used to store information but are not used for querying.

Extend Legacy API and Improve Database Reliability

Description

- Reduce maintenance costs by migrating data from an MSSQL database hosted on EC2 instances to Aurora Serverless. Build a facade API in front of the legacy API to hide the latter's

Problem

- The project team received requirements from the Client to optimize the maintenance and infrastructure costs of an existing MSSQL database hosted on EC2 instances.
- Also, the existing legacy API did not return all the necessary data for a new frontend application.

Solution

- To address the database problem, Intetics suggested migrating data to the AWS Aurora Serverless database. There was a requirement from the Client to use the PostgreSQL engine, as they already have experience with it.
- To address the legacy API problem, Intetics suggested building a facade API using API Gateway with Lambda services instead of updating the legacy API. This approach also keeps existing legacy API as-is and will not affect existing integrations.

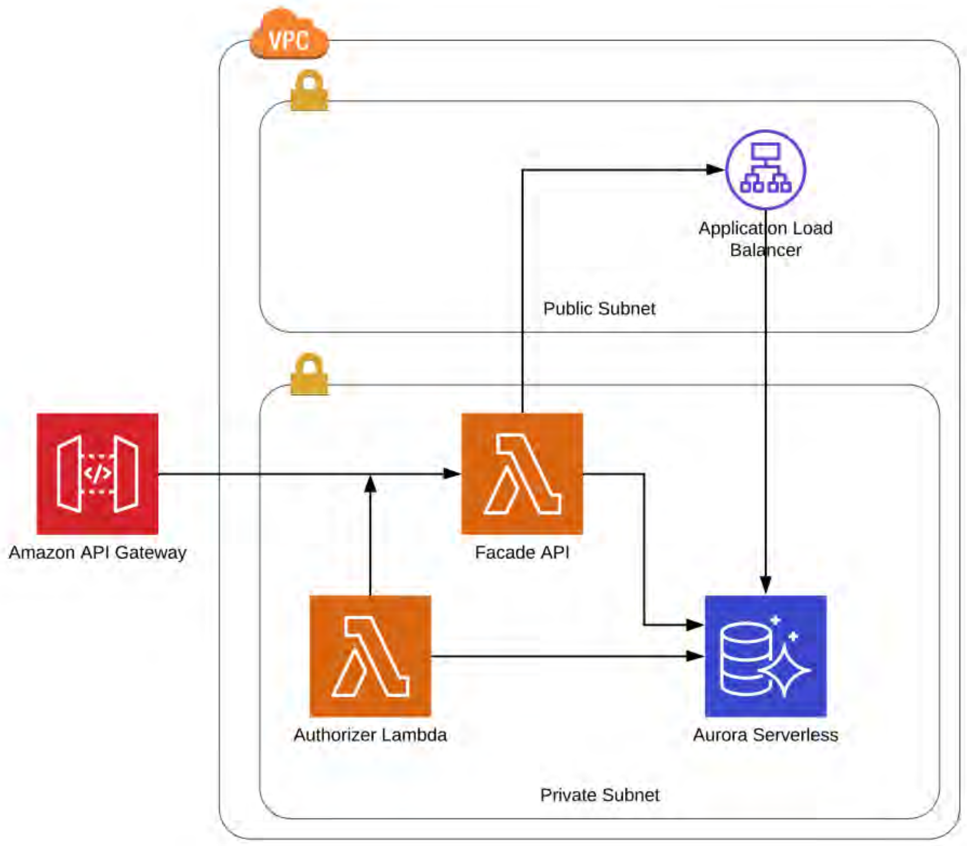
Outcome

- The Aurora Serverless database reduces the cost of database maintenance and allows the team to focus on building business features.
- Facade API provides all necessary data that allows building a new frontend application.

Lesson Learned

- Using fully managed services is the key to improving Time-to-Value.

Solution Details



The solution contains the following components:

- The API Gateway is integrated with Facade Lambda. The solution uses the private REST API Gateway. An edge-optimized endpoint type is used for better performance, as API requests are routed to the nearest CloudFront Point of Presence.
- Facade Lambda is used to hide the complexity of legacy API and provide all necessary data in a single response. Lambda itself reads data from the legacy API and from the Aurora Serverless database.
- Authorizer Lambda is used to protect the facade API and to verify legacy tokens from requests that are stored in Aurora Serverless. The API Gateway Lambda authorizer is used to verify that only users with enough permission have access to call this API.

API Gateway

- The team uses a throttling feature for the API Gateway to prevent our API from being overwhelmed by too many requests.
- Also, caching was enabled for endpoints that return static data. TTL for the cache was configured for 1 hour, as this data can periodically be changed (it was discussed with the Client).
- Latency, Request Count, and 4xx and 5xx Error Count are used to determine the health and status of the API.

Aurora Serverless Database

- Short summary:
  - Initial size of the database – 2 GB
  - Expected yearly growth of the database – 100 MB
  - Number of tables/objects in the database – 50
  - Anticipated number of concurrent requests during peak database usage – 200
  - Database engine used in the application – PostgreSQL
- Previously, the Client ran the MSSQL database on EC2 instances. Intetics replaced EC2 instances with Aurora Serverless. The same subnets and security groups were used.
- Migration to RDS reduced operational costs and helped to review the application and optimize performance.
- From load test results, the RDS instance with PostgreSQL works faster than EC2 with MSSQL.
- UAT was performed by the Client in the Staging environment before release.

- The application was verified by the QA team. Load testing was performed. Results were provided and discussed with the Client.
- Route53 was used to create a DNS name for the database endpoint.
- Flyway was used before to define database schema. The team didn't use any specific features from MSSQL, and the difference was mostly in data types (BIT -> BOOLEAN, VARCHAR(max) -> TEXT, etc.).
- The application used the ORM framework, which works fine with MSSQL and PostgreSQL databases.

  The release plan included the next steps:

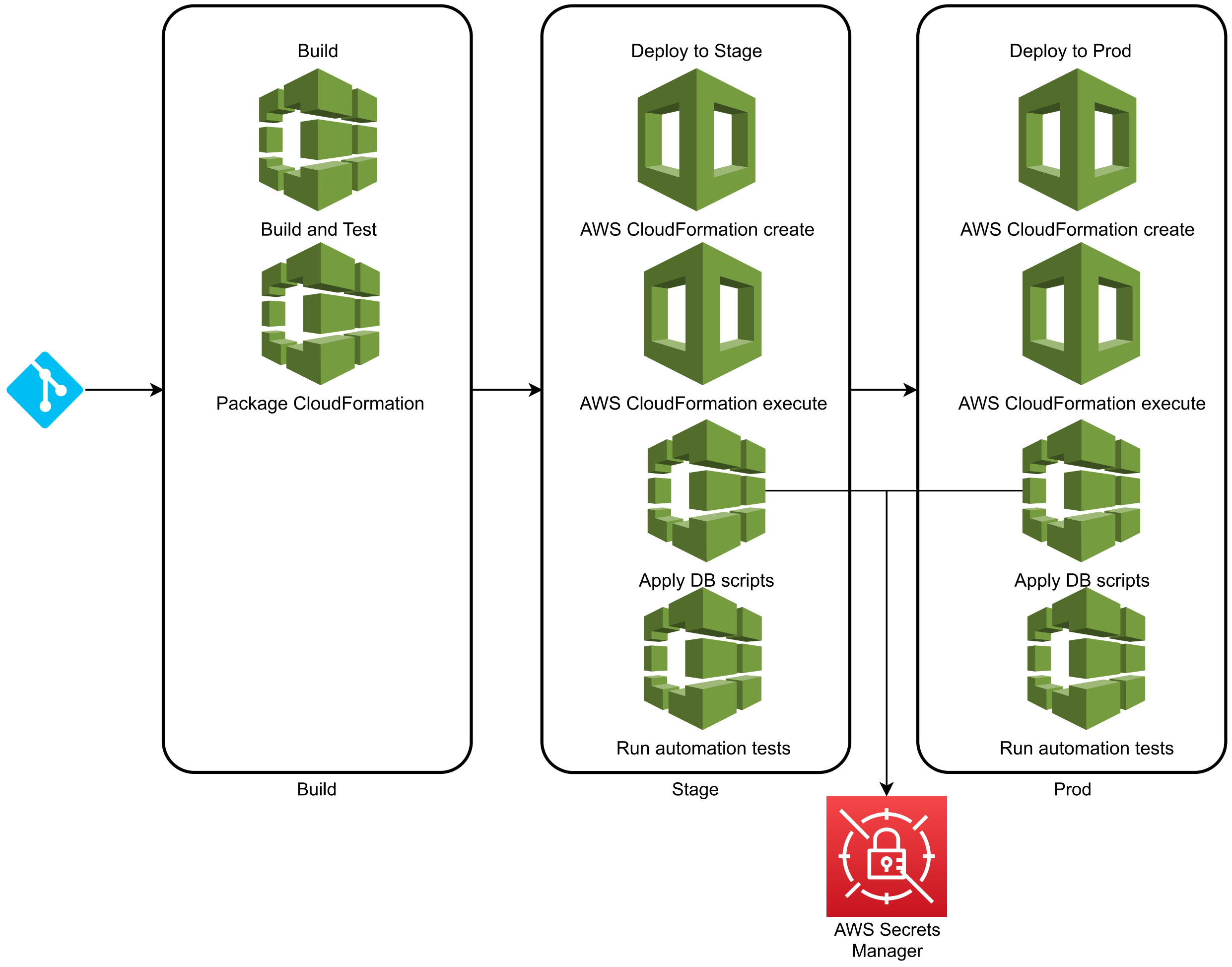
  1. Migrate all the data 1 day before release.
  2. Stop all the traffic.
  3. Migrate delta of data in the database.
  4. Release an updated application that uses the new database
- The rollback plan included the next steps (for cases in which issues are found within 24 hours of release):
  1. Release a previous version of the application that uses the old database. Customers accept potential data loss.
  2. If an issue is found after 24 hours of release – then a hotfix should be prepared and released.
- Applications that work with the Aurora Serverless database declare two data sources: first for writing and reading "critical" data and second for read-only operations. It allows using different endpoints for read and write operations.
- CloudWatch metrics are used to monitor the Aurora Serverless state.
- The team has JMeter load tests for the application, which show application endpoints with long response times. The team runs this test before every release to see changes in the application's performance. For slow endpoints, the team adds caching where possible or does profiling on the application and optimizes logic and database queries.
- CloudWatch Event Rule with clean-up Lambda is used to clean up obsolete data within the database.
- As a long-term solution, the Client wants to migrate data to DynamoDB where possible.
- To continuously improve TCO, the team implemented an automatic mechanism to clean up the database's obsolete data and back up historical data to S3 and Glacier. Also, load testing highlights places to optimize (via adding a cache or by updating logic or queries).
- Confluence pages were created with details for database schema changes.
- When adding new columns or tables, the development team should discuss them with the Client, as well as with their Data Protection Officer.

Deployment Process

- Git is used as a version control system to store source code (including CloudFormation code and buildspecs for CodeBuild) for this solution.
- CloudFormation is used to describe AWS resources.
- CodeBuild is used to build the application, run unit and integration tests, and do static code analysis.
- CodePipeline is used for Continuous Deployment.

Roll Back Process

- Versioning for Lambdas is used to keep the previous version. It allows for rolling back particular Lambda in case of issues after release.
- The current solution has Automation tests that verify API.
- Functionality from Lambda functions is covered with unit and integration tests.
- Infrastructure is described using CloudFormation. CodePipeline is used to apply CloudFormation on Stage and Prod environments.
- Flyway is used to describe database schema. CodeBuild is used to apply DB scripts from CodePipeline.
- Stage and Prod environments are created in the same AWS account. Each environment has its own VPC, subnets, and security groups. Separate credentials are used per environment.
- IAM policy is used to restrict application access for credentials between environments (for example, Stage env can access only RDS credentials for Stage env).
- IAM policy allows human access to credentials for all environments. An SQL script is prepared to anonymize the PII column on the Stage environment.
- The CodeDeployDefault.OneAtATime deployment configuration used to have zero downtime during deployment.



Application Monitoring

- Default CloudWatch metrics (for Lambda, API Gateway, RDS, åand CodePipeline services) cover all requirements for metrics for the current solution.
- The CloudWatch Dashboard is used to visualize how the solution works.
- CloudWatch Alarms are configured with defined thresholds.

- For API Gateway, the team uses Latency, Request Count, and 4xx and 5xx Error Count to determine the health and status of the API.
- For RDS: CPU usage, database connections, and free storage space metrics are used for alarms.
- All logs from Lambdas are available in CloudWatch logs. Intetics uses the INFO log level.
- X-Ray is configured for all Lambdas.
- CloudWatch Alarms configured on API Gateway: Latency, Request Count, and 4xx and 5xx Error Count are used to determine the health and status of the API.

Security

- All Lambdas are added to private subnets in VPC.
- AWS services were used as IAM principals: application-autoscaling.amazonaws.com, lambda.amazonaws.com, and apigateway.amazonaws.com.
- The current solution uses only RDS to store data located in private subnets. Database credentials are rotated by the AWS Secrets Manager. RDS itself is encrypted.
- Database credentials are stored in the AWS Secrets Manager service, which rotates them every 90 days.
- Fine-grained IAM policies are used to restrict access to credentials. 2FA is required to retrieve credentials for human access.
- CloudTrail logs contain information about who accessed database credentials.
- Log information about the database is captured in CloudTrail logs and Aurora Serverless database logs.
- The current solution doesn't have any automated way to analyze logs for potential security events.
- Aurora Serverless is used from the internal network only. It's located in VPC, inside private subnets.

Security Groups

» For this solution, Intetics has a few types of Security Groups: for CodeBuild, an individual security group for VPC Lambda, security groups for EC2 instances, and security groups for RDS.

» Security Groups for Lambdas and CodeBuild:
  - has no inbound rules
  - allows all traffic in outbound rules

» For EC2 instances in Auto Scaling Groups, Intetics has individual Security Group per ASG with:
  - inbound rules: allows http access (80 port) for ALB (using ALB Security Group Id for Source)
  - outbound rules: allows all traffic

» Security Groups for RDS:
  - allows all internal traffic in the 10.0.0.0/8 IP range for the 5432 port.
  - allows all traffic in outbound rules.

Auditing

» CloudTrail is enabled in all AWS Regions.

» CloudTrail log file integrity validation is enabled.

» CloudTrail log files are stored on S3 on a separate AWS account.

» Versioning is configured on S3 with logs.

Improve Security and Performance for a Platform

Description

- Improve a platform’s performance and data reporting by protecting the site from crawler requests.

Problem

- The Client has a publicly available site, and different crawlers scan it. This produces an undesirable load on the platform and affects reporting data.
- Static content on S3 is not protected.

Solution

» To address the first issue, the team built CloudFront in front of the core platform to accelerate access to data. The AWS WAF service was utilised. The following rules were configured:
  - XSS/SQL injection blocking rule
  - Rate Limits rule
  - User-Agent blocking rule

» To address the second problem, the team build Authorizer@Edge using Lambda@Edge. It verifies JWT tokens and allows access to static content from S3 only for authorised requests.

Outcome

- 15% of the traffic produced by crawlers is now blocked automatically.
- The site performance improved, and there are no outages caused by scans from crawlers.

TCO Calculation

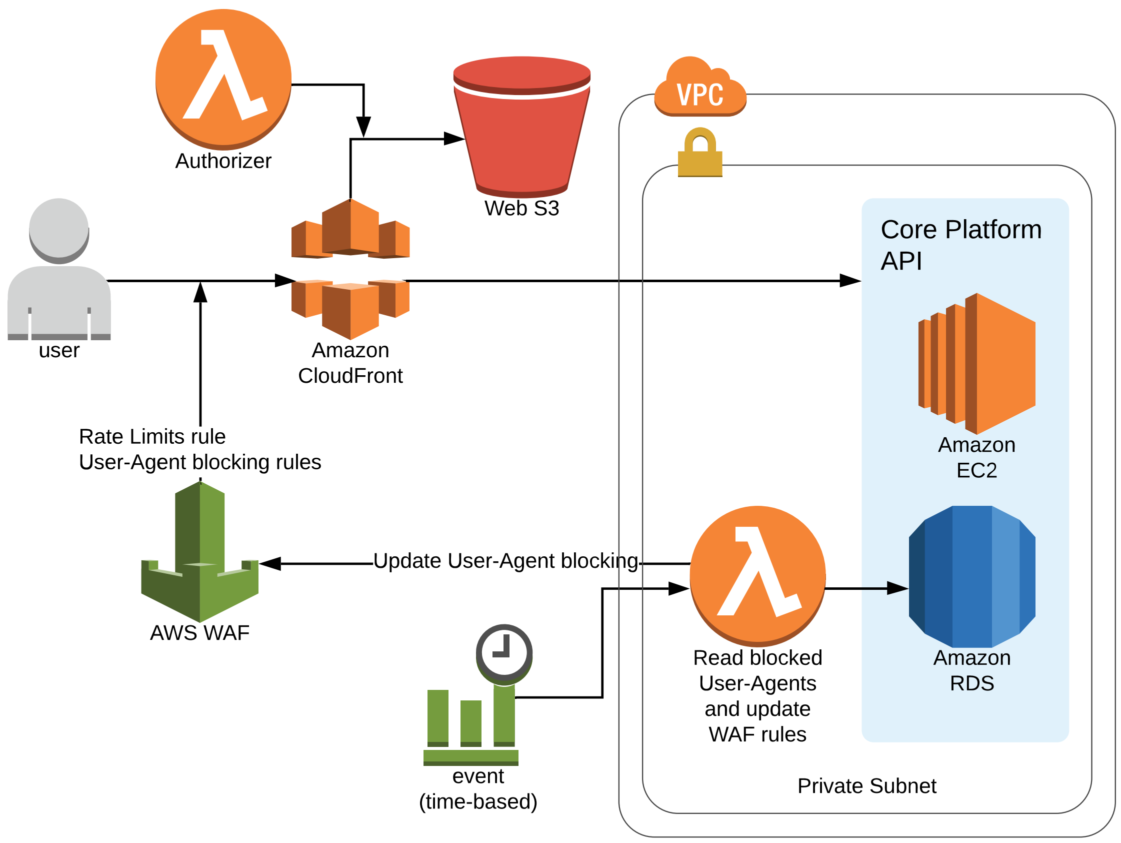
- The expected TCO was calculated based on:
  - AWS WAF Web ACLs
  - Requests to CloudFront
  - Requests to S3 files and authorizer Lambda invocations

  Summaries of estimates were provided and discussed with the Client before implementation.

Lesson Learned

- AWS provided a full set of tools to protect the public site.

Solution Details



- The Client has an application that is used by users all over the world.
- To improve the performance, availability, and security of delivering content, Intetics utilised CloudFront.
- The solution used two origins:
  - Web S3 with static content (images, frontend applications, etc.)
  - ALB, which provides dynamic and static content from the Client core platform
- The cache was configured on CloudFront for static data from S3 and from ALB.
- CloudFront Origin Security Features:
  - HTTPS-only connection enforced between CloudFront and ALB origin

Lambda@Edge

- Authorization@Edge was implemented using Lambda@Edge to allow only authorised access to S3 content. Users include the JWT token in a content request, and Authorizer Lambda verifies if that user (JWT) can access the resource.

AWS WAF

- AWS WAF used for the following purposes:
  - The Web ACL rule for XSS/SQL injection attacks was configured to protect the Java web application from such attacks.
  - Rate limits are configured to protect the platform from "unsupported" crawlers. Different people or systems run scripts that scan the site for information and produce a huge load on the platform. To protect the platform from being overwhelmed by such crawlers, the team configured this rule.
  - Block user-agents – most crawlers use their own user-agent. Intetics has a mechanism that can find which user-agents are from crawlers. The team uses this rule to block requests with such user-agents.
- To protect platforms from bots/crawlers, the following rules are configured:
  - The Rate Limits rule was configured to limit requests from the same IP address
  - The User-Agent blocking rule was configured to block requests from User-Agents that are identified as bots.
  - All page view events are tracked on the platform. The reporting team builds reports for end clients with views and statistics. That reporting tool can define potential crawlers based on user-agents from request headers. The reporting team saves such user-agents in a separate table.
  - Intetics has an AWS Lambda that is periodically triggered by the CloudWatch Event Rule and updates the WAF rule based on user-agents from the database.

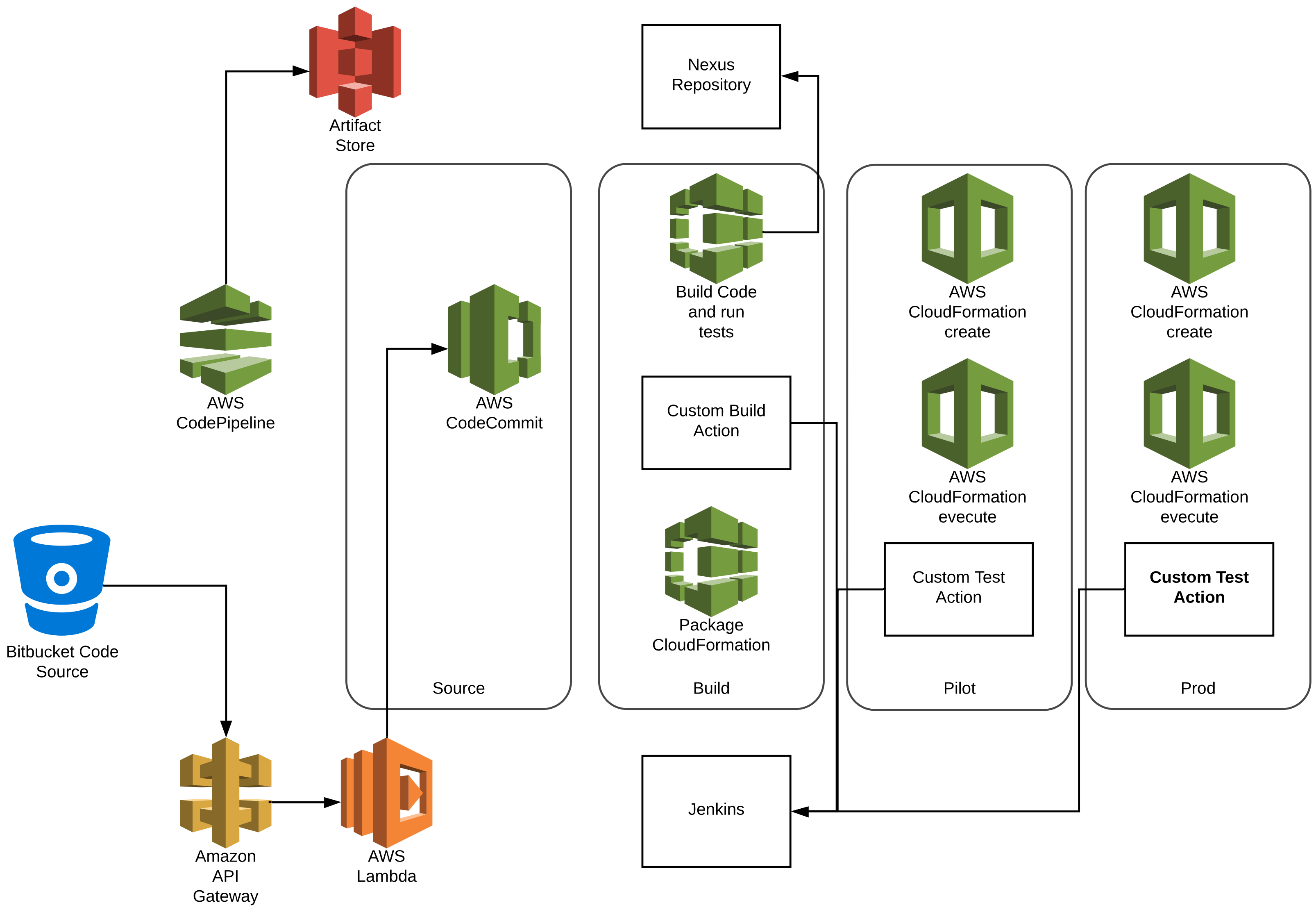
- WAF implementation improved the site's performance, as requests from most crawlers are blocked. It also improved views reporting, as it now includes only data from real users.
- XSS and SQL injections and blocking by user-agent rules are verified by automation tests.
- Rate Limits rules are verified by running JMeter load tests.

Deployment Process

- The source code is hosted on Bitbucket. The solution with API Gateway and Lambda is used to sync changes from Bitbucket to CodeCommit.
- CloudFormation is used to describe AWS resources.
- CodeBuild is used to build a Java application, run unit and integration tests, and do stylechecks.
- CodePipeline is used for Continuous Deployment.
- Jenkins, with a SeleniumBase test, is used to run automation tests against the pilot and prod environments. EC2 instances with Jenkins also contain SonarQube.

Roll Back Process

- Versioning for Lambdas is used to keep the previous version. It allows for rolling back particular Lambda if there are issues after release.
- The solution is fully covered with automation tests, and it helps to find issues in the pilot environment – and to release to production only verified and stable changes.


Artifact Store
Nexus Repository
AWS CodePipeline
AWS CodeCommit
Source
Build Code and run tests
Custom Build Action
Package CloudFormation
Build
AWS CloudFormation create
AWS CloudFormation evecute
Custom Test Action
Pilot
AWS CloudFormation create
AWS CloudFormation evecute
Custom Test Action
Prod
Bitbucket Code Source
Amazon API Gateway
AWS Lambda
Jenkins

Application Monitoring

- All logs from Lambdas are available in CloudWatch logs. Intetics uses the INFO log level.
- X-Ray is configured for all lambdas.
- CloudWatch Alarms are configured on CloudFront 4xx and 5xx errors and request latency; Lambdas – Error count and Duration.
- Default CloudWatch metrics (for Lambda, CloudFront, and CodePipeline services) cover all requirements for metrics for the current solution.
- CloudWatch Dashboard is used to visualise how the solution works.
- CloudWatch Alarms are configured with defined thresholds.
- CloudWatch Dashboard is used to visualise how the solution works.
- CloudWatch Alarms are configured with defined thresholds.

Security

- The solution includes AWS WAF to protect the site. The AWS Lambda authorizer is used to restrict access to static content from S3.
- AWS services were used as IAM principals: lambda.amazonaws.com and cloudfront.amazonaws.com.
- Security Groups for RDS:
  - allows all internal traffic for the 10.0.0.0/8 IP range for the 5432 port.
  - allows all traffic in outbound rules
- VPC lambda has the following Security Groups rules:
  - no inbound rules
  - allows all traffic in outbound rules

- "Redirect HTTP to HTTPS" on CloudFront is configured.
- An HTTPS-only connection is enforced between CloudFront and ALB origin.
- The current solution only used data from RDS. Credentials for RDS are stored in the AWS Parameter Store.

Auditing

- CloudTrail is enabled in all AWS Regions
- CloudTrail log file integrity validation is enabled.
- CloudTrail log files are stored on S3 on a separate AWS account.
- Versioning is configured on S3 with logs.

Public Cloud Processing

Description

- A solution to retrieve, process, and visualize a client’s public cloud usage data.

Problem

- The Client has a number of clients that run their platforms on public clouds.
- There was no centralized place where each client could see complete information about their spending on infrastructure and usage of cloud resources.

Solution

- The proposed solution is based on AWS' fully managed services and contains the following components:
  - Credentials API – REST API to specify how/where to retrieve reports about cloud resource usage.
  - Report Uploader – integrates with public cloud services to retrieve and save information for further processing.
  - ETL processing – processes information from downloaded reports.
  - Reporting API – REST API to retrieve aggregated information about used resources and visualize this information.
  - Abstracted API – REST API to hide complexity on integration with external Rest APIs
- The proposed solution was built using Lambda, DynamoDB, Aurora Serverless, Time Series, API Gateway, SSM, CloudFormation, CloudWatch, X-Ray, CodeBuild, CodeDeploy, CodePipeline, SNS, SQS, and StepFunction AWS services.

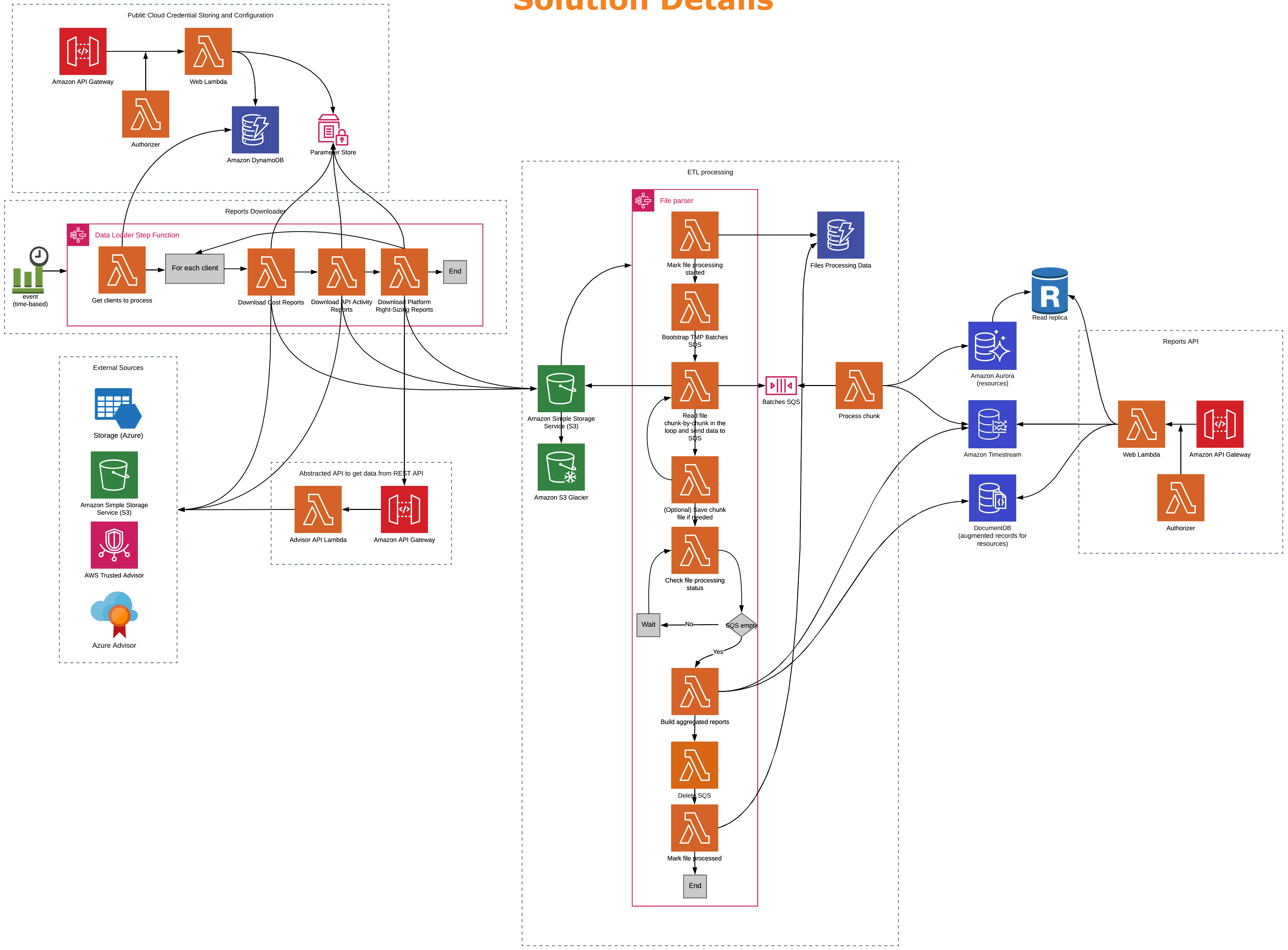
Outcome

- The Clients got a single place to view and manage their cloud resources.
- The solution can be easily extended to support other public cloud providers or integrated with other vendors.

Lesson Learned

- Fully managed services can significantly reduce operational costs when running the platform in test environments.

Solution Details



- The platform contains the following components:
  - Credentials API
  - Report uploader
  - ETL processing
  - Reporting API
  - Abstracted API
- The whole solution was split into small, isolated components to simplify development, maintenance, and testing processes.

Credentials API

- It’s an API Gateway + Lambda-based REST API.
- API will allow for creating, updating, deleting, getting a configuration, or getting all existing configurations for the client.
- DynamoDB is used to store data. Entities that are stored there include:
  - Client: ClientId, Name
  - Credentials: CredentialId, ClientId, Access, CreatedDt, UpdatedDt
  - Configuration: ConfigurationId, ClientId, CredentialId, ReportType, Config (JSON object with all the necessary data)
- The cache is disabled on the API Gateway.

Report Uploader

- This Lambda-based solution will periodically check external sources for new information and will upload new information to S3 for further processing.

- The CloudWatch Event Rule is used to trigger the scanning mechanism.
- Step Function is used to orchestrate Lambdas uploads. Step Function contains the next steps:
  - The initial step is to read the current clients and their configurations from storage.
  - After that, it runs upload logic for each client in the loop:
    - Upload ReportA Lambda – read credentials from DynamoDB and upload the file to S3.
    - Upload ReportB Lambda – read credentials from DynamoDB SM and upload the file to S3.
    - Upload ReportC Lambda – read credentials from DynamoDB and upload the file to S3.
- If needed, Step Function can be reconfigured to run the upload mechanism independently for each report for each client on the defined Cron expression.

ETL Processing

- This component contains elements that will process downloaded CVS files and will save results to storage.
- S3 bucket is used to store downloaded files. File names should include client id, report type, and datetime. Additional information can be defined via metadata if required.
- To optimize costs, the lifecycle rule is configured to archive files to Glacier after N days.
- Versioning and cross-region replication are disabled on S3.
- ETL processing is built using Lambdas. Step Function is used to orchestrate Lambdas.
- DynamoDB is used to store information about processed files. It helps with troubleshooting and will provide details on how ETL works.
- TTL is configured on DynamoDB to clean up old records.
- When a new CSV file is added to S3, Step Function will be automatically triggered and will run ETL for this file.
- Step Function has the following steps:
  1. Mark file processing as started Lambda – based on file name and metadata, add information about the file to "Files Processing Data" storage.

2. Bootstrap TMP Batches SQS lambda – creates a temporal SQS queue with a unique name for current file processing. Later chunks of data will be sent to SQS for further processing. SNS will allow for decoupling read files and processing data processes, and in this way, the team can process data in parallel using multiple Lambdas.
3. Read the file chunk-by-chunk in the loop and send data to the SQS lambda. S3 allows for reading specified range bytes from the object. This Lambda will read a file from S3 chunk-by-chunk in the loop and send data to SQS in batches (N rows per event).
4. Check the file processing status Lambda. After the whole file is read and all data is sent to SQS for further processing, Step Function will wait until all data is processed.
5. Build reports Lambda – augmented reports will be built based on data from Amazon Timestream and saved to DynamoDB.
6. Delete SQS Lambda – clean up temporary SQS after data is processed.
7. Mark file processed – update the current file processing information in "Files Processing Data" storage.

» The Process chunk Lambda will process data from SQS. SQS events should contain metadata about the report type and client. This Lambda will:
  - Read data from SQS
  - Filter and pre-format data
  - Save all data to Amazon Timestream
  - Save resource information to Amazon Aurora

Reporting API

» It's an API Gateway + Lambda REST API. API enables the reading of reporting data and resource details.

» A cache will be used on API Gateway.

Abstracted API to Get Data from REST APIs

» Abstracted API should encapsulate all logic of reading data from different sources and provide a unified API to get that information. It will simplify the process of downloading reports from REST APIs and will allow it to easily be extended to other reports that are available by REST API.

» The API itself is API Gateway + Lambda-based REST API.

» The cache is disabled on API Gateway.

In addition, each component will have:

- Its own repository with code and CloudFormation templates.
- A CI/CD pipeline built using CodeBuild and CodePipeline services.
- CloudFormation would be used to describe infrastructure as a code.
- AWS SAM is used to describe serverless resources.
- API Gateways will have a dummy Authorizer Lambda. They will be used later to integrate with the existing Auth solution.
- Encryption is enabled for all storage (SQS, RDS, S3, Glacier DynamoDB, and Timestream).
- All applications should be covered with unit and integration tests.
- X-Ray will be enabled for resources. It will be used for troubleshooting.
- Default CloudWatch metrics will be used for monitoring. All custom CloudWatch metrics won’t be used.
- CloudWatch Dashboards will be created for each component of the platform.
- CloudWatch Alarms with email notifications will be used for degradation and outage alarms. Details on the threshold should be discussed and agreed upon.
- CloudWatch logs will be used to store logs from Lambdas, API Gateways, and CodeBuild.
- Each component will have its own Playbook with information about infrastructure, as well as guides on how to monitor and troubleshoot issues.
- X-API-KEYS will be used for API Gateways.
- The platform will be created for Dev, Stage, and Prod environments.
- An AWS account exists. The platform will re-use existing resources (VPC, subnets).
- All components will be covered with automation tests.
- ReservedConcurrency will be configured for Lambdas.

Deployment Process

- Git is used as the version control system to store the source code (including CloudFormation code and buildspecs for CodeBuild) for each component of this solution.
- CloudFormation is used to describe AWS resources. The AWS Serverless Application Model (AWS SAM) is used as a CloudFormation extension to get reliable deployment capabilities. AWS CloudFormation templates are split into logical stacks, so modules are decoupled, reusable, and easier to maintain.
- Templates also take advantage of nesting and/or cross-stack references.
- The CloudFormation validate-template command is used to validate the template at the build stage of CodePipeline
- CodeBuild is used to build an application, run unit and integration tests, and do static code analysis.
- CodePipeline is used for Continuous Deployment.

Roll Back Process

- Versioning for Lambdas is used to keep the previous version. It allows for rolling back to a particular Lambda if there are issues after release.
- The current solution has Automation tests that are integrated into CodePipeline and verify the application in all environments after the new version is deployed.

Application Monitoring

- CloudWatch Dashboard is used to show the overall status of the solution. Default CloudWatch metrics (for Lambda, DynamoDB, RDS, SNS, Step Function, and CodePipeline services) cover all requirements for the current solution's metrics.

- CloudWatch Dashboard is used to visualize how the solution works.
- CloudWatch alarms are configured to send degradation and outage events in case of errors.
- CloudWatch Alarms configured on Lambdas - Error count and Duration; Step Functions - failed executions.
- CloudWatch logs and X-Ray are used to investigate issues.
- All logs from Lambdas are available in CloudWatch logs. Intetics uses the INFO log level.
- X-Ray is configured for all lambdas.

Security

- Pylint is used for static code analysis. It's performed as part of the Build step in AWS CodeBuild.
- Credentials for the root user are accessible to only one person from the Client team. IAM Users are created for an Intetics team member who needs access to the AWS account. Access keys are not assigned to the root user. MFA is enabled for the root user as well as for IAM Users.
- All Lambdas are added to private subnets in VPC.
- AWS services were used as IAM principals: application-autoscaling.amazonaws.com, lambda.amazonaws.com, and sns.amazonaws.com.

Security Groups

- For this solution, Intetics has three types of Security Groups: one for CodeBuild, individual security groups for each VPC Lambda, and security groups for EC2 instances.

- Security Groups for Lambdas and CodeBuild:
  - Have no inbound rules
  - Allow all traffic in outbound rules

Auditing

- CloudTrail is enabled in all AWS Regions.
- CloudTrail log file integrity validation is enabled.
- CloudTrail log files are stored on S3 on a separate AWS account.
- Versioning is configured on S3 with logs.

Streaming Platform Development

Description

- A platform that offers functions for ordering physical hardware, streaming, and providing recording history, as well as social features.

Problem

- The Client developed a hardware device that can connect to multiple cameras and produce a live show, for the purpose of streaming events. They needed a platform that could provide a full set of functionality—from ordering physical devices to streaming live video and supporting stream recording history.
- Additionally, the platform should provide social features (comments, likes, chats, etc.).

Solution

Intetics based the proposed solution on the following aspects:

1. **Ease of Use for Users, Administrators, Site Planners, Editors, Translators, and Developers**
   Users will be provided with the best user experience in the industry. Based on our extensive experience with web, desktop, and mobile applications, and our extensive portfolio of similarly scaled solutions, the team will deliver an intuitive, fast, customizable user interface. For device owners, Intetics will provide a dedicated space with detailed instructions. They will have a simple device registration form, and they also will be able to configure the device and manage streams from one place. All user roles will enjoy a superior experience.
2. **Flexibility**
   Applying reliable tools with various existing components like Angular and Bootstrap will allow us to build pre-defined templates for the stream pages. Elements will be able to have different positions on the page. Users will be able to upload cover images and choose themes. Many pre-defined Bootstrap themes will be available to apply.
3. **Scalability**
   The suggested approach uses AWS fully managed services that can scale in a second and handle an almost unlimited number of streams and viewers.
4. **Technology**
   The solution includes the latest cloud technologies and approaches. For UI, a micro-frontend approach will be used with a modern JavaScript framework. The extended list of proposed technologies is given in the System Architecture section.
   Having a close relationship with Amazon, Intetics has access to their technical support at all stages of development. Amazon Web Services (AWS) provides over-the-top (OTT) live video streaming solutions to cost-effectively deliver media content to a global audience in the AWS Cloud. AWS is used by top live-streaming services like Twitch, Amazon Prime Video, etc. Amazon also supports startups.
5. **Cost and revenue models**
   Potential paid features are paid subscriptions, premium streams, donations, a marketplace with plugins/addons, a paid site constructor, and ads.

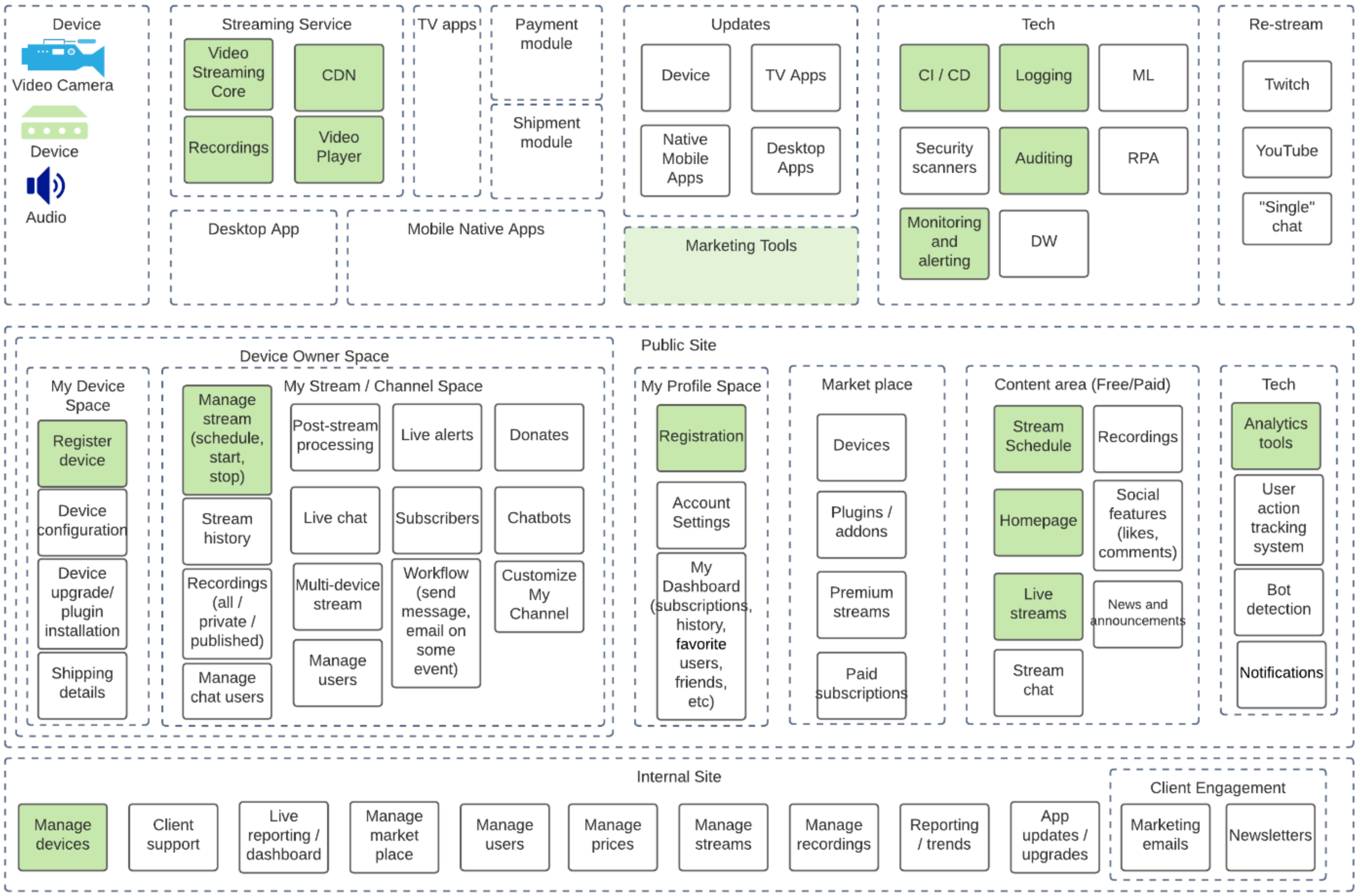
Outcome

- The Client received a platform that allowed for streaming live videos, recording and storing streams, tracking user actions, engaging with other users, and more.

Lesson Learned

- The AWS Serverless approach allows us to build a scalable platform that can handle an almost unlimited number of streams and viewers, and at the same time, AWS will charge the platform only for used resources and data.

Solution Details



Green blocks relate to the high-priority scope that was covered in the first phase.

Block Name	Description
User action tracking system	Tracking user actions on the site, shipping data to Data Warehouse for further analysis, reporting, and providing recommendations
Bot detection	–
Notifications	–
Internal Site	Site area and related functionality to manage the platform
Manage devices	–
Client support	–
Live reporting/ dashboard	–
Manage the marketplace	–
Manage users	–
Manage prices	–
Manage streams	–
Manage recordings	–
Reporting/trends	–
App updates/upgrades	–
Client Engagement	The module that adds the ability to engage new users (content consumers, venues, artists) in an easy way
Marketing emails	–
Newsletters	–

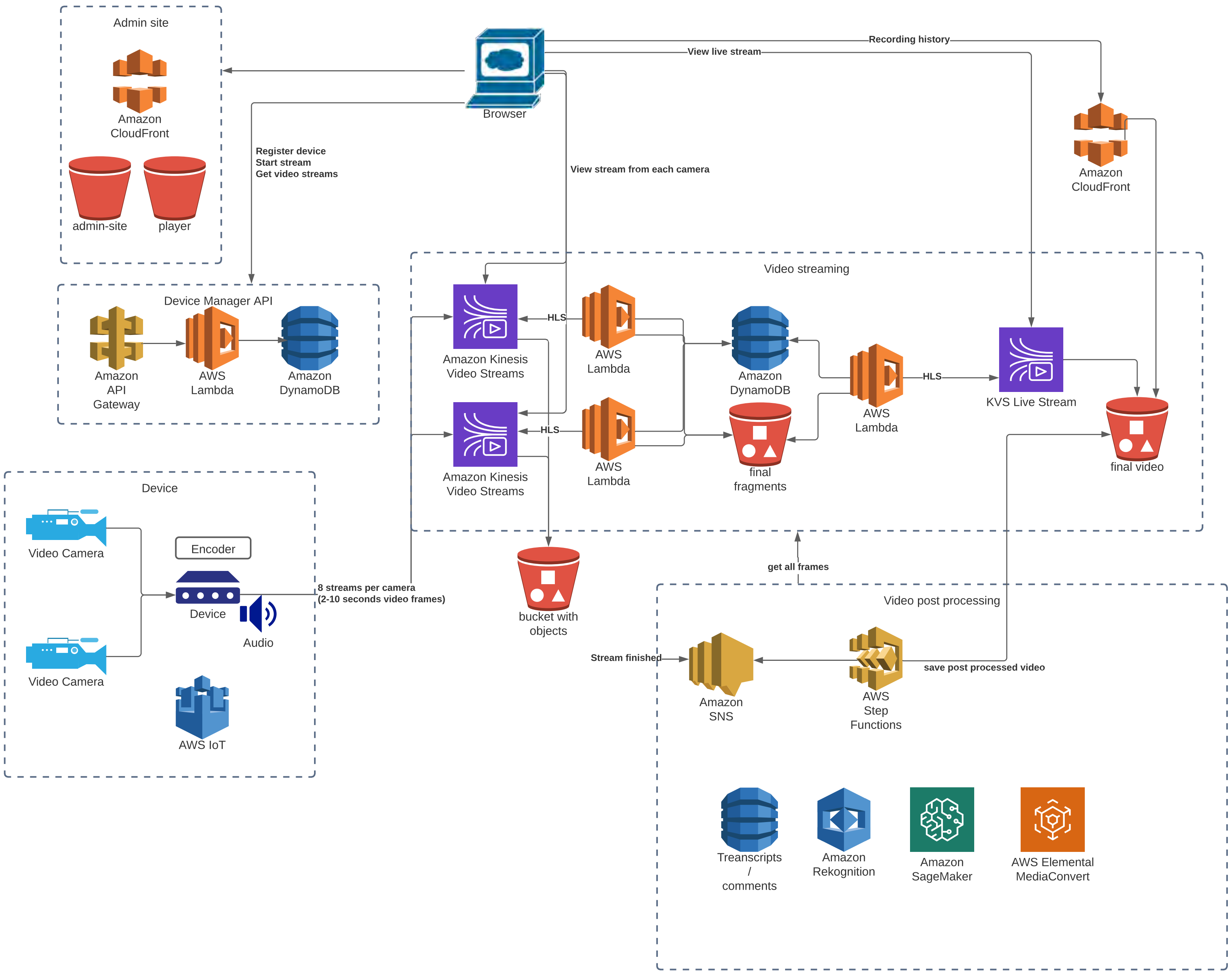
Block Name	Description
Native Mobile Apps	–
TV Apps	–
Marketing Tools	Technical means/tools/services for marketing activities. As an example, Sparkpost, Marketo, and Oracle Responsys
Tech	–
CI/CD	Continuous integration and continuous delivery
Logging	Application-level logging
Security scanners	Configure vulnerability scanners
Auditing	An audit allows the removal of unneeded IAM users, roles, groups, and policies, and it makes sure that users and software have only the permissions that are required
Monitoring and alerting	To maintain the platform, it is recommended to have detailed monitoring metrics and automatic alerts on degradation or outage platform events
Data Warehouse	The system that will store all the data from the platform and that will be used for reporting and analytics
Machine Learning	Machine Learning to solve various tasks (for example, to recognize nudity or to filter chats)
RPA	Robotic process automation for routines
Re-stream	Integrate with other streaming platforms
Twitch	Re-stream video to Twitch
YouTube	Re-stream video to YouTube
“Single” chat	Accumulate chat messages from all platforms
Public site	The publicly available site that can be used by Artists and Viewers
Device Owner Space	Part of the site where Device Owners can register and configure their device and also manage streams and recordings
My Device Space	Device-related functionality
Register device	Register the device as simply as possible

Block Name	Description
Device Configuration	–
Device upgrade/plugin installation	–
Shipping details	–
My Stream/Channel Space	Area for device owners to work with their content
Manage stream (schedule, start, stop)	–
Stream history	–
Recordings (all/private/published)	Including custom video uploads
Manage chat users	–
Post-stream processing	–
Live chat	–
Multi-device stream	–
Manage users	–
Live alerts	–
Subscribers	–
Workflow (send a message, send an email about some event)	–
Donates	Ability to donate to Artists or Venues
Chatbots	Additional way to interact with users
Customize My Channel	Ability to upload a custom cover image for the “My Channel” page and choose pre-defined styles. It will contain a constructor where end users can choose how to display functional blocks on the page

Block Name	Description
My Profile Space	Account-related functionality
Registration	–
Account Settings	–
My Dashboard (subscriptions, history, favorite users, friends, etc)	–
Marketplace	–
Devices	–
Plugins/addons	Custom extensions for the device: for example, color correction filters
Premium streams	–
Paid subscription	–
Content area Free/Paid)	–
Stream Schedule	–
Homepage	–
Live streams	–
Stream chat	–
Recordings	–
Social features (likes, comments)	–
News and announcements	–
Tech	Module to enhance user’s interaction
Analytics tools	Like Google Analytics, etc.

Block Name	Description
User action tracking system	Tracking user actions on the site, shipping data to Data Warehouse for further analysis, reporting, and providing recommendations
Bot detection	–
Notifications	–
Internal Site	Site area and related functionality to manage the platform
Manage devices	–
Client support	–
Live reporting/ dashboard	–
Manage the marketplace	–
Manage users	–
Manage prices	–
Manage streams	–
Manage recordings	–
Reporting/trends	–
App updates/upgrades	–
Client Engagement	The module that adds the ability to engage new users (content consumers, venues, artists) in an easy way
Marketing emails	–
Newsletters	–

High-level architecture of the system



Technology stack

Currently, Intetics focuses on the AWS Serverless approach. It allows us to build a scalable platform that can handle an almost unlimited number of streams and viewers and, at the same time, AWS will charge the platform only for used resources and data.

Purpose	Technology, Tool
Public Cloud Provider	AWS
Provisioning	Infrastructure-as-a-code approach
	AWS CloudFormation/Terraform
Monitoring	AWS Cloud Watch
	PagerDuty
	Pingdom
Runtime	Docker
	AWS ECS
	AWS Lambda
Storage	AWS S3
	AWS EFS
CDN	AWS CloudFront
Language and Frameworks	Java, Spring or .Net
	Python, Flask
	Angular or React
	Swift, Kotlin
AWS Cost Management	AWS Budget
	AWS Cost Explorer

Purpose	Technology, Tool
Data	AWS DynamoDB
	AWS RDS
	AWS Neptune
	AWS ElastiCache
CI/CD	Git
	AWS CodeBuild
	AWS CodeDeploy
	AWS CodePipeline
Application Integration	AWS Step Functions
	AWS SNS
	AWS SQS
Customer Engagement	AWS SES or Sparkpost
	Twilio
	Oracle Responsys
Logging	AWS CloudWatch
	Splunk
Tracing	AWS CloudTrail
	AWS X-Ray

Purpose	Technology, Tool
Security	AWS IAM
	AWS Cognito
	AWS KMS
	AWS WAF
	AWS Shield
Management and Governance	AWS Config
	AWS System Manager
	AWS Trusted Advisor
	AWS Organization

Purpose	Technology, Tool
Analytics and Reporting	AWS Athena
	AWS Redshift
	AWS Kinesis
	AWS QuickSight
	AWS Glue
	Google Analytics
	Tableau
Payment	Stripe/PayPal

intetics

Where software concepts come alive™

INTETICS MEANS YOUR SUCCESS

Toll Free: +1 (877) SOFTDEV
US: +1 (239) 217-4907
DE: +49 (211) 3878-9350
UK: +44 (20) 3514-1416
Email: intetics@intetics

www.intetics.com