Management Tools in AWS

Analytics Tensor

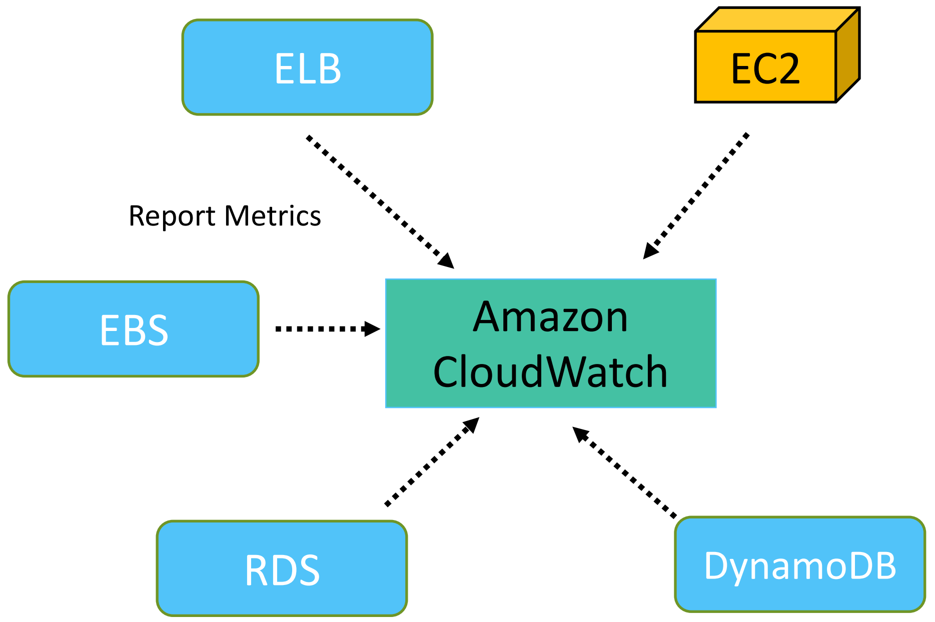
Mahesh KC

mahesh.kc@analyticstensor.com

https://analyticstensor.com

Amazon CloudWatch

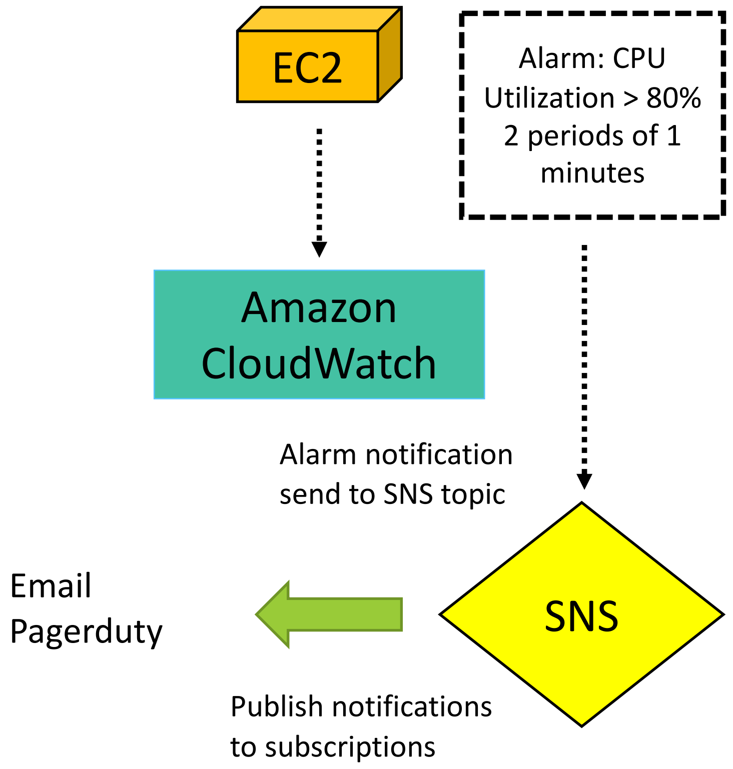
- Amazon CloudWatch service is used to collect metrics from our resources such as EC2, Elastic Load Balancer, EBS volume, RDS, DynamoDB and more.
- CloudWatch is used for collecting metrics. The metrics is used for 2 weeks. If we want to compare logs from more than two weeks then we can use CloudWatch API to pulls the metrics.
- Each services has its own unique collection of metrics. For e.g. RDS, EC2, EBS has its own collection of metrics.
- We can also publish custom metrics into CloudWatch. We have to pay for certain metric. For e.g. If we want to check total number of processes running in EC2, then we can write a code and send the metrics to CloudWatch. We can also have application level metrics too.
- EC2: The default interval for EC2 to collect metrics is 5 minutes. If we want to change to 1 minutes then have to pay per instance. EC2 only report the information those are only available to hypervisor such as CPU utilization, Network I/O, Disk I/O. It doesn't capture memory metrics out of the box. We have to write our self.
- ELB: Default is 1 min interval. It capture such as number of request, number of http 200, 300, 400, and 500 responses, connection error etc.
- RDS: It measure the memory usage. It also measure the number of connections, how the disk is used.
- DynamoDB: It measure number or read/write throughput.



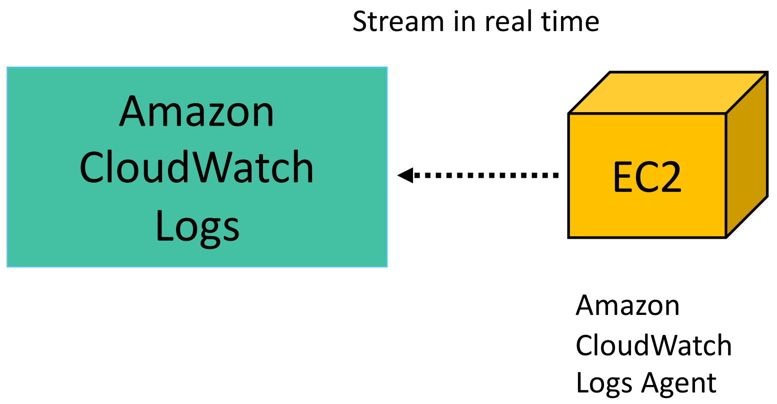
Amazon CloudWatch is a metrics repository.

Amazon CloudWatch Alarms

- Amazon CloudWatch not only stores the metrics but also created alarms based on those metrics.
- The alarms is triggered when it reaches to the maximum threshold. For e.g. we have EC2 running with some application, then we want to be alerted when the CPU utilization reaches above 80% more than 2 minutes, it is 2 periods of 1 minutes. After there we can perform number of other task such as send the alarm notification to SNS topic, which will publish the notification to different subscription either email or pagerduty. Pagerduty is a service that send alert to cell phone.
- The is a limit in services. We get up to 5000 alarms per account. We can increase by creating support ticket to AWS.
- There are three possible states of alarm:
  - OK: Every thing is OK.
  - ALARM: The metric is above or below what we defined in alarm.
  - INSUFFICIENT_DATA: We don't know yet. i.e. when we create alarm at the beginning and the state is unknown since it hasn't collected any data or doesn't have sufficient information.
- Some resources doesn't report value of 0. For e.g. in the SQS if the message count is zero then it won't report. It will flag as insufficient data.


EC2
Alarm: CPU Utilization > 80% 2 periods of 1 minutes
Amazon CloudWatch
Alarm notification send to SNS topic
SNS
Email
Pagerduty
Publish notifications to subscriptions

Amazon CloudWatch (cont.)



Amazon CloudWatch Logs

- With Amazon CloudWatch Logs we can stream our logs in real time to the CloudWatch Log services.
- We need to install Amazon CloudWatch Logs Agent to our EC2 instances, configure it to watch the logs files written into disk. It can be anything like application logs, Apache logs etc. All the logs file will be be stream to Amazon CloudWatch Logs.
- The default retention is indefinite. We can also decrease it.
- We can archive those logs into Amazon S3.
- We can also stream those logs to Amazon Elasticsearch and creates dashboard using Kibana for searching information stored in Elasticsearch.
- Install CloudWatch Logs Agent: https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/install-CloudWatch-Agent-on-EC2-Instance.html
- In CloudWatch log we can also search and filter log data. We can search using specific syntax. It can also search JSON fields.
- We can create subscription filters stream to Kinesis and Lambda. i.e. When the particular log come then it will filter and goes to specific services.
- We can also create metric filters. For e.g. count the metrics. From the count we can create alarm. For e.g. when we have more than 20 request of 400. Count bytes transferred. Count customer conversion. i.e. when customer closes shopping cart in applications.

For e.g. Filter the pattern for ERROR UNKNOWN gives match such as:

[ERROR] Network Issue
[ERROR] Exception handled on class A.
[ERRPR] IllegalArgumentException

Filter and Pattern Syntax:
https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/FilterAndPatternSyntax.html

Infrastructure as Code

- Infrastructure as Code (IaC) is the management of infrastructure in a description model. We can think the infrastructure as code to configure the infrastructure by writing either script or CloudFormation templates.
- Architecture is be complex. There are lot of moving part such as VPC, subnet, route table, network access control list, security group, EC2 with various types and size, database, DynamoDB table etc. Performing all manually is always challenging. Creating resources manually have several issue such as:
  - Reliability: We need to know how we created based on our design.
  - Reproducibility: If we want to take the same architecture and create on several environment i.e. Dev/QA/Prod or Other AZ.
  - Documentation: Creating documentation take a time and some information can be opted.
- Challenges with scripts: We can create using bash shell scripts with aws cli tools, python boto libraries. Some issues are:
  - Dependencies: Creating some resource should be defined first in order than others. For e.g. we need to create EC2 instances in VPC, we need to create subnet. Before creating subnet we need to create VPC. The script should know all of those.
  - Parallelization: Some resources can be created in parallel.
- All of the problems described above can be solved using AWS CloudFormation.
- With CloudFormation we can have:
  - Template: JSON-formatted text file with described resources. We can put the file into source control.
  - AWS CloudFormation Engine: We can upload the file in Cloud Formation Engine. This engine will processes template then create, update, and delete resources. The engine knows the order of how the resources need to be created, we don't have to worry about it. But if there is ambiguity such as launching database and EC2 instance, we can control the order.
  - Stack: Collection of resources created by template. Once the resource is created we can manage them.

Amazon CloudFormation

- Amazon CloudFormation provide a language to model and provision AWS and third party application resources in cloud environment.
- Let's watch the video: https://aws.amazon.com/cloudformation/
- Amazon CloudFormation is free when we provide the the resources with AWS::*, Alexa::*, and Custom::* namespaces.
- It is declarative programming compared to imperative programming. Imperative programming is concerned with specifying all the steps one after other, we are telling how to do and what to do. But with declarative programming, we are just declaring the ways we want, the CloudFormation Engine will automatically build on the order so we don't have to worry about steps.
- The template is a JSON file so it can store in source control.
- Templates can be nested. We can nest the template by putting one template inside other template.
- It is write once, deploy many.
- We can create library for commonly deployed architectures.
- AWS CloudFormation offers access to full breadth of AWS.
- Template is created using JSON and we'll declare following in template:
  - Parameters: Variables
  - Resources: Resource to be created such as VPC, RDS.
    - Properties:
    - Conditions:
  - Mappings: Map one value to other value.
  - Outputs: Control the output of the template. e.g. create load balance and get the attribute back as output from load balancer.

Template Anatomy:
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/template-anatomy.html

Example of Templates:
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-sample-templates.html

Amazon CloudFormation (cont.)

Template Sample:

https://aws.amazon.com/cloudformation/templates/aws-cloudformation-templates-us-west-1/

Let some example for resource such as S3, EC2, and other application stacks.

S3:https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-s3-bucket.html

```
{
        "AWSTemplateFormatVersion" : "2010-09-09",
        "Description" : "AWS CloudFormation to create S3_Bucket: It create a publicly
        accessible S3 bucket.",
        "Resources" : {
                "S3Bucket" : {
                   "Type" : "AWS::S3::Bucket",
                      "Properties" : {
                         "BucketName": "analyticstensor-2020-04",
                          "AccessControl" : "PublicRead"
}}},
"Outputs" : {
   "BucketName" : {
       "Value" : { "Ref" : "S3Bucket" },
         "Description" : "Create publicly accessible bucket named analyticstensor-2020-
04. "
}}}
```

Template to create publicly accessible S3 bucket

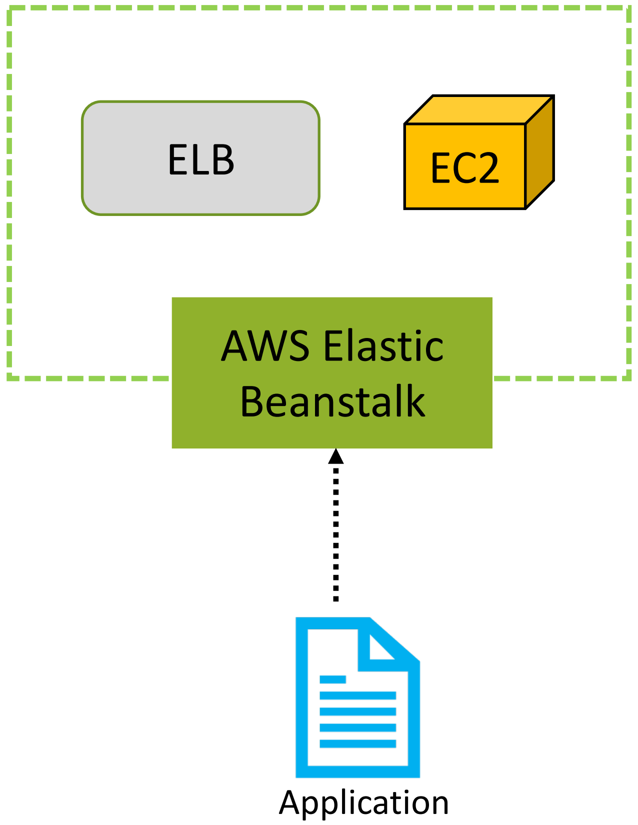
Analytics Tensor

Demo: Amazon CloudFormation (cont.)

- Choose CloudFormation. Click “Create Stack” in right corner. Choose “with new resources”.
- We can template on various way:
  - Template Designer: Drag an drop UI tool in AWS.
  - Other Tool such as: VisualOps
- Choose create template in designer option.
- Click create template in designer.
- Drag and drop S3 Bucket from resource panel.
- Rename the bucket to “yournameyyyymmdd”.
- Click File Icon in left top corner. Click save either on local file or s3 bucket. Let’s choose local file. OR choose create stack (cloud icon) next two component right to file icon.
- We can also provide IAM role. Under Advance option, we can specify the rollback configuration (i.e. if the resources fail then trigger alarm using CloudWatch. We can also send notification. Under stack creating, we can specify roll back on failure to enable or disable, this means when resource fail then do we need to rollback or leave where it fails. (i.e. last point the resources that it failed.)
- Click next. Enter “create-aws-bucket” in stack name. Choose next and click create stack.

Amazon Elastic Beanstalk

- It is an easy-to-use service for deploying and scaling web applications and services developed with Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker on Apache, Nginx, Passenger and IIS server.
- We can upload the code and Elastic Beanstalk with automatically handles the deployment, from capacity provisioning, load balancing, auto-scaling to application health monitoring.
- There is no charge for Elastic Beanstalk, we have to pay for the AWS resources for storing and running applications.
- Elastic Beanstalk is an application management platform that provides very easy entry into AWS.
- It is ideal for developers. Developer normally want to write the application and make it run. They don't want to handle resources such as install and manage resources like Apache server etc. Developer can upload the application and Beanstalk will handle it.
- When application is uploaded in Beanstalk, the Beanstalk service will handle the creating of resources and manage the resources automatically.
- When uploading the application we can choose: load balancing, auto scaling, monitoring, and capacity provisioning then Beanstalk will automatically handles.



Demo: Amazon Elastic Beanstalk

- Choose Elastic Beanstalk from Services located in Compute Category..
- Click Applications on left panel and choose “Create a new application”.
- Provide application name “My-First-WebApps”.
- Click Create.
- Click Environment. Choose Web server environment.
- Under Platform, Choose Managed Platform.
- Choose Python. This will create Django WebApps.
- Choose simple application.
- Click Create create environment.
- You site should be running.
- Check the URL and share with other for testing.