Neptune.io: Architecture

Neptune is fully hosted on AWS. It uses Amazon EC2 for compute, DynamoDB for our database layer. It is currently hosted on 3 data centers in AWS us-east-1 (Virginia East) region. All of our internal services are architected to gracefully handle single data center wide outage. This not only applies to compute but also applies to database layer since AWS replicates the data across multiple availability zones in the region. We also leverage Amazon SQS for queuing services and SES to send emails to our customers to ensure high availability. Neptune services have a SLA of 99.9%, except for scheduled maintenance hours.

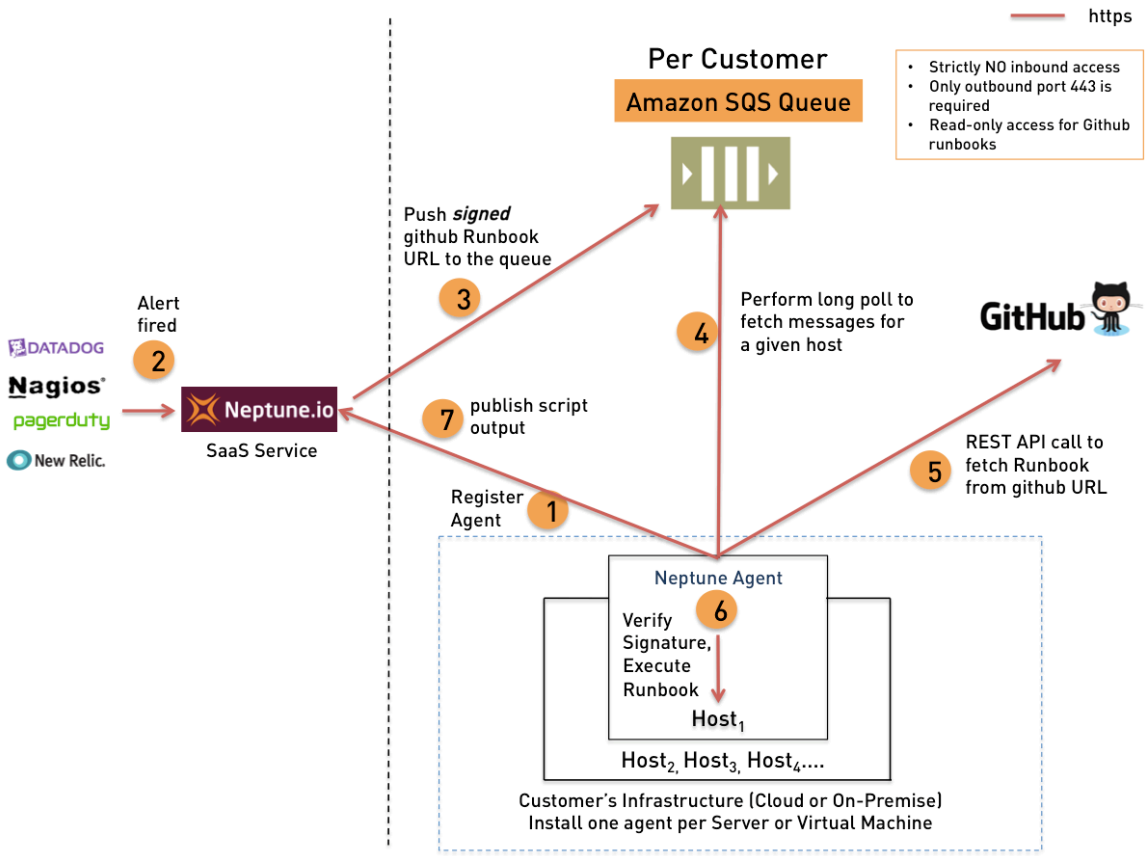

Figure 1. Neptune Agent Architecture

First, the document provides a brief overview of various components involved and then it describes each of the steps in architecture diagram in much more detail.

Components:

Monitoring/alerting tools: These tools generate alerts and you need to do one-time integration with these tools by following the appropriate integration guides.

Neptune SaaS Platform: Neptune SaaS platform is the highly available remediation-as-a-service platform that orchestrates all the activities with various components.

Amazon SQS Action Queue: There will be a dedicated action SQS queue per customer, thus giving isolation among other customers. All the Neptune agents will poll this queue for any messages.

Neptune agent: A piece of software written in Go, with no software dependencies that sits on customer's servers. The agent uses SQS long polling technique to poll for any messages from the SQS queue, and if a message is destined for a particular agent, then it executes the action on the host or VM and reports the results back to Neptune SaaS platform.

Github Runbook Repository: This provides full-control to Customers in terms of "what runbooks can be executed" on Neptune agents. This means only agents will only run the runbooks specified in the github repository and nothing else. Also, no one else other than customer will have the ability to modify/add new runbooks. It further ensures that runbooks are reviewed and vetted before you start using them. Neptune will only have read-only access to the repository.

Below we'll describe each of these steps in more detail.
(Note that all the communication happens over https)

Step1: In this step, you install Neptune agent on all your servers where you need to perform corrective actions. The agent registers with the Neptune SaaS platform and gets a unique id for itself. It also starts heart beating with Neptune periodically if the registration is successful. Agent doesn't require any inbound port access; it only requires http port 443 to be open for outbound access.

Step2: This is the entry point for sending alerts into Neptune. As a pre-requisite, you need to do one-time integration with your monitoring and alerting tools (i.e. API keys) to send all the relevant alerts into Neptune. These API keys are encrypted using a secret key before storing it on our database. This ensures that even if Neptune AWS keys were to be compromised, an attacker will not have access to your monitoring tool API keys unless the attacker also compromises encryption secret key, which is stored and managed separately. This step ensures that as soon as the alert is fired from any of these monitoring tools, Neptune will pick the alert if the integration is setup correctly.

Step3: Neptune's rule engine then parses the incoming alert and identifies the appropriate action and pushes the signed github URL into an customer specific Amazon SQS Queue. Github URL and signature serves two different purposes. Github URL ensures that only a

runbook specified in github repo can be executed at the agent. Signature ensures that it was sent by Neptune only. If an attacker gets hold of Neptune AWS keys, he would be able to push the message the SQS queue but he can't replicate the Neptune's signature because he doesn't have Neptune's private key. Unless the private key and Neptune's AWS keys are compromised, he can't execute any github runbook on customer's servers. Note that private key and AWS keys are managed and secured separately at Neptune.

Step4: All Neptune agents poll the SQS queue for any new messages. If there is a message destined for a specific host (which can be found by looking at the message body after the signature is verified), then that agent will pick up that message for further processing. Neptune agent will delete the message after the message is processes.

Agents authenticate to SQS queue using temporary AWS STS tokens. These temporary access tokens get rotated every 4 hours, and when the token is expired, Agent authenticates to Neptune using an API key and fetches a new token. Refer to AWS STS (http://docs.aws.amazon.com/STS/latest/APIReference/Welcome.html) and temp creds usage (http://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_temp.html) for further reference. Note that agent, at best, can only read or delete messages from SQS queue, but the agent doesn't have permission to write new messages into the SQS queue. This will also ensure that a compromised agent on a single box can't create unnecessary messages for other machines.

Step5: In this step, agent verifies that message signature and ensures that it's sent by Neptune SaaS' platform before acting on it. Once verified, it pickup the github URL for the runbook from the message body. After that, it talks to github REST API using a read-only-api key and fetches the actual runbook script. Agent will only act if it gets 200 response from github API call. Note that this is done on-demand so that we always pick up the latest from github repo.

Step6: Agent now executes the runbook script on the host or VM, and captures the output of the script.

Step7: The output and exit status of the script are then sent to Neptune via https REST API call. This ensures the exit status or timeout is captured correctly in Neptune web UI.

Note: *This architecture describes SaaS model. If for some reason, SaaS model doesn't work for you, we do offer on-premise dedicated AWS VPC deployment model. Reach out to us at support@neptune.io for more information.*