NGINX Plus on AWS

Scott Ward

October 2014

Abstract

Amazon Web Services (AWS) provides a reliable, scalable, secure, and highly performing infrastructure for the most demanding web applications. AWS offers infrastructure that matches IT costs with customer traffic patterns in real time. In turn, NGINX Plus brings end-to-end performance and scalability to modern web architectures built on top of AWS, while alleviating the burden of heavy lifting of HTTP for application developers and system engineers. NGINX Plus is built on top of the open source NGINX web server, and offers additional features around load balancing, monitoring, proxy routing, and advanced management of the NGINX configuration.

This whitepaper provides specific technical guidance on how to deploy and configure NGINX Plus on AWS. Additionally, we outline key integrations and configurations that are unique to AWS products and allow flexibility around using the NGINX Plus product to best meet your needs.

Introduction

Organizations need to ensure that they can keep up with rapid changes that can be required for their global computing infrastructures. Additionally, they need to find ways to deploy and deliver applications from distributed, cloud-based services with confidence that these applications can deliver a consistent and reliable level of service, withstanding huge and unpredictable spikes in traffic without missing a beat. The compute resources offered by AWS meet this need by providing a global computing infrastructure as well as services that simplify managing that infrastructure.

Based on the NGINX open source web server software, NGINX Plus is a high performance, high concurrency HTTP reverse proxy, load balancer, edge cache and origin server—all in one compact, software-only package. NGINX Plus is architected for multiple advantages:

Scalability and Performance—NGINX Plus scales with the resources of the Amazon EC2 instance that it is deployed on: CPU resources for SSL, memory and disk for caching, and network for bandwidth. There are no built-in limits on its performance.

Capability—NGINX Plus offers features such as reverse proxy with load balancing, application request routing, content acceleration, and caching. Additionally, NGINX Plus offers many opportunities to integrate with existing AWS products.

Predictability—NGINX Plus was built to minimize the use of CPU and memory resources. This design allows NGINX Plus to run effectively on Amazon EC2 instances with a small footprint and low overall infrastructure costs.

Usability—NGINX Plus setup is flexible, logical, and scalable. Readability and manageability of NGINX Plus configuration saves time and reduces errors.

Automation—NGINX Plus configuration can be automated with tools like Puppet and Chef.

Monitoring and Logging—NGINX Plus offers activity monitoring with performance metrics exposed via JSON. NGINX Plus also offers standard logging capabilities and remote logging support via the industry-standard syslog protocol.

AWS Services Overview

Throughout this whitepaper, there are references to AWS services and their usage with NGINX Plus. This section provides an overview of each service.

Amazon EC2

Amazon Elastic Compute Cloud (Amazon EC2) provides resizable compute capacity in the cloud. When drawing a comparison between traditional infrastructures, Amazon EC2 represents the servers that run your applications, web servers, and databases.

Security Groups

Amazon EC2 security groups act as a firewall governing network traffic in and out of EC2 instances. AWS customers have full access to define and assign security groups that are appropriate for their EC2 instances. By default, EC2 instances are launched with security groups that are in an allow nothing state for inbound traffic. Changes to allow the appropriate inbound traffic to the EC2 instance must be made by the customer.

Elastic IP Address

An Elastic IP address is a static IP address designed for dynamic cloud computing. Elastic IP addresses can be assigned to an individual EC2 instance and then re-mapped to another EC2 instance to replace the original instance in case of a failure or a migration. Elastic IP addresses can only be assigned to one EC2 instance at a time.

Auto Scaling

Auto Scaling allows you to scale your Amazon EC2 capacity up or down automatically, according to conditions that you define. Auto Scaling allows the number of EC2 instances in use to seamlessly increase during demand spikes to maintain performance, and decrease automatically during demand lulls to minimize costs.

Amazon Elastic Block Storage

Amazon Elastic Block Store (Amazon EBS) provides persistent block-level storage volumes for use with EC2 instances in the AWS cloud.

AWS Identity and Access Management

AWS Identity and Access Management (IAM) enables you to control access to AWS services and resources securely. Using IAM, you can create and manage AWS users and groups and use permissions to allow and deny their access to AWS resources.

Amazon CloudWatch

Amazon CloudWatch is a monitoring service for AWS cloud resources and applications that run on AWS. CloudWatch provides functionality to view graphs and statistics for the metrics that are monitored. These metrics can be viewed individually or combined with other local metrics, or metrics from other instances, to provide a complete picture.

Additionally, CloudWatch can create alarms, based on metrics, which send notifications or take additional actions such as initiating Auto Scaling actions.

Elastic Load Balancing

Elastic Load Balancing automatically distributes incoming application traffic across multiple EC2 instances in the cloud. It enables you to achieve greater levels of fault tolerance in your applications, seamlessly providing the required amount of load balancing capacity needed to distribute application traffic.

AWS Command Line Interface

The AWS Command Line Interface (AWS CLI) is a unified tool to manage your AWS services. With just one tool to download and configure, you can control multiple AWS services from the command line and automate them through scripts.

Amazon Route 53

Amazon Route 53 is a highly available and scalable Domain Name System (DNS) web service. Amazon Route 53 effectively connects user requests to infrastructure running in AWS, such as EC2 instances, Elastic Load Balancing load balancers, or Amazon S3 buckets, and can also be used to route users to infrastructure outside of AWS.

Where NGINX Plus Fits

The **Web Application Hosting in the AWS Cloud: Best Practices** whitepaper highlights the following components of a classic web application architecture to leverage AWS infrastructure:

App Server Load Balancer. Software load balancer on EC2 instances to spread traffic over the app server cluster.

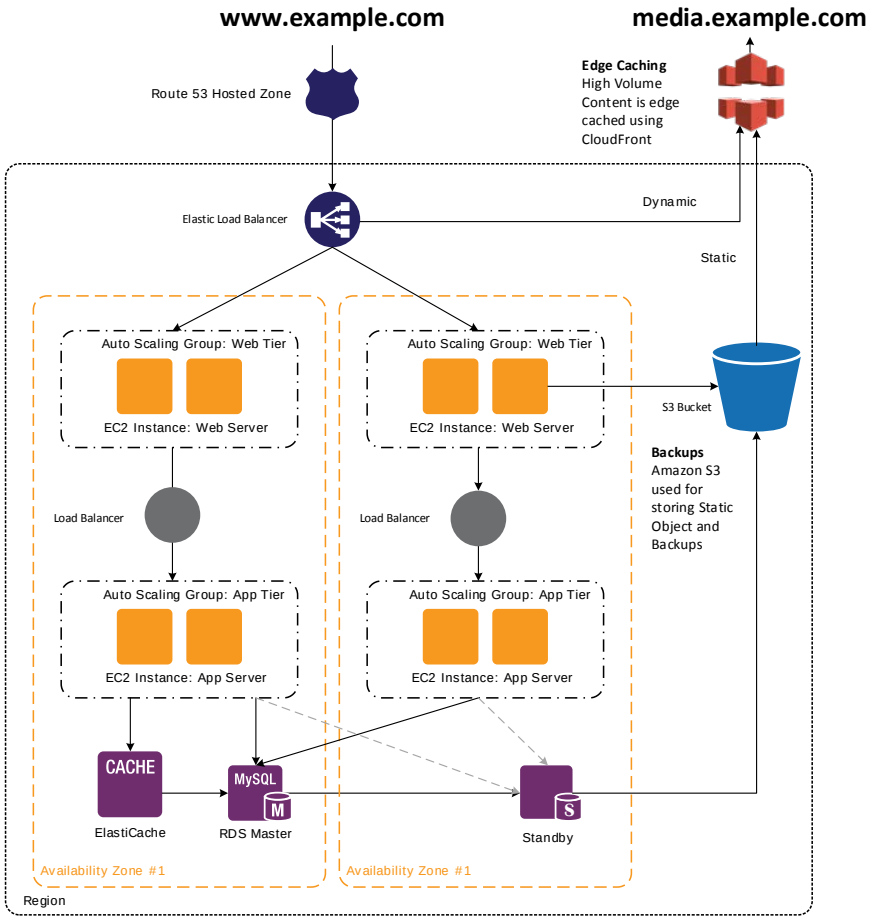
Auto Scaling Web Tier. Group of EC2 instances handling HTTP requests.

Auto Scaling App Tier. Group of EC2 instances running the actual app. Instances belong to the Auto Scaling group.

Amazon ElastiCache. Provides caching services for the app, removing the load from the database tier.

Streaming Media Server. Provides capabilities for streaming media applications, including progressive downloads, pseudo-streaming with MP4 and FLV, and adaptive streaming for video-on-demand (VOD) applications.

NGINX Plus provides quite a few essential web infrastructure tools that fit into the above-mentioned applications. Different features in NGINX Plus can be combined and configured for simultaneous use on the same instance. Alternatively, NGINX Plus can be set up in accordance to the functionality of a particular tier, without compromising either performance or scalability.



Using NGINX Plus on Amazon EC2

Basic Installation

With AWS, you can create and launch one or more EC2 instances running NGINX Plus.

The recommended and often easiest way to get started is to use the NGINX Plus AMIs, which are available in the AWS Marketplace. These AMIs provide the latest version of NGINX Plus for use with AWS, pre-packaged software for NGINX Plus configurations, and helper scripts for installing and configuring typical development environments with NGINX Plus.

To get started, visit the AWS Marketplace and search on “NGINX Inc”. Select the NGINX Plus AMI that you want to use. When prompted, sign in to your AWS account.

On the AMI details page, review the details about the AMI. Click **Continue** when you are ready to start the process of creating an EC2 instance and installing NGINX Plus.

One-Click Launch from the AWS Marketplace

If you prefer not to use the Amazon EC2 console to launch your instance, you can configure all the details about your instance on the **One-Click launch** tab from the AMI details page. This option allows you to launch only one EC2 instance at a time.

1. Select the **One-Click launch** tab from the AMI details page.
2. In the Region section, select the AWS region where you want to launch your NGINX Plus instance.
3. In the Amazon EC2 Instance Type section, choose your preferred instance type.
4. Choose the VPC that your instance will be created in.
5. Review the proposed security group settings and either choose from your existing security groups or accept the new proposed security group included in the AMI definition.
6. In the Key Pair section, choose a key pair to associate with the instance. This key is used to connect to your instance over Secure Shell (SSH).
7. Click **Accept Terms & launch with 1-click** to create your new instance.

Launch with the Amazon EC2 Console

If you prefer to use the Amazon EC2 console to launch your instance, click the **Launch with EC2 Console** tab from the AMI details page. This option allows you to launch multiple EC2 instances.

1. Select the **Launch with EC2 Console** tab from the AMI details page.
2. Click **Accept Terms** to allow access to the Amazon EC2 console.
3. Choose your instance type.
4. Configure the instance details including the number of instances, VPC, and monitoring.
5. Add any additional storage.
6. Tag your instance.
7. Configure the appropriate security group for your instance.
8. Launch your instance.

Connect to and Verify Your Instance

1. After the instance has been created, you are presented with the deployment details. Take note of the instance ID.
2. Sign into the AWS Management Console, select Amazon EC2, and click the ID for the instance just created. In the instance detail tab, make a note of the Public DNS hostname. You need this to log in to your NGINX Plus instance.
3. Connect to the EC2 instance via SSH. For more information about how to connect to an EC2 instance, see Connect to an Instance in the *Amazon EC2 User Guide for Linux*.
4. Upon new instance launch, NGINX Plus starts automatically and is configured to serve a default *index.html* page. You can verify this configuration by placing the public DNS of your EC2 instance into a browser and confirming that the default NGINX page is returned. On your EC2 instance, you can also check the status of NGINX Plus by running the following command:

```
$ /etc/init.d/nginx status
```

NGINX Configuration Files

This whitepaper references multiple configuration options for NGINX Plus along with configuration snippets. Unless otherwise referenced, the configuration file referenced in this whitepaper is **/etc/nginx/nginx.conf**. The nginx.conf file is the base configuration file for your NGINX instance. Through the include statement, which is a pointer to files in other directories, the nginx.conf file may reference additional configuration files which may be relevant to your configuration needs.

For more information about how to configure NGINX Plus, see the Beginners Guide section on NGINX.org.

Performance

When configuring infrastructure to support your application or service, it is important to take the time to plan for how your infrastructure will perform under expected and un-expected loads. Planning for performance involves not only picking the correct infrastructure but also configuring your infrastructure to flex up or down when needed, ensuring that your instances are configured for performance, and that you use tools that allow you to test how your infrastructure performs under load.

Amazon EC2 Instance Sizing

When it comes to sizing your EC2 instance for NGINX, it's not possible to give prescriptive rules on sizing for a user's application. Every application, NGINX Plus configuration, and profile of user traffic is different.

Many users first deploy NGINX Plus on small instances for simple development environments. These small development environments are well served by the Amazon EC2 T2 instance class. When it comes to the first production deployment, many users select general-purpose EC2 instances such as m3.medium or m3.large. If traffic levels grow, users have the option of scaling vertically with a larger EC2 instance or horizontally using Auto Scaling.

Some traffic profiles have particular requirements. If you're terminating SSL connections, then CPU resources are a must, and NGINX scales linearly in performance with the CPU capacity of the EC2 instance that you deploy it on. Similarly, if your traffic is dominated by small requests and WebSockets requests, CPU is a key resource.

If you are caching content on an EC2 instance, then disk I/O is a bottleneck. NGINX provides hints to the operating system page cache to keep hot content in memory, but disk writes and cache misses affect performance. When you are looking to use NGINX for content caching, select EC2 instance types that offer SSD instance storage and get the fastest possible access to information that must be retrieved from disk. In regards to the size of the cache you can operate, this is dictated by the memory capacity of the EC2 instance that is chosen to host the cache.

If your workload is bandwidth-heavy—for example, if you are serving video files or large downloads—then network bandwidth may be your limiting resource and horizontal scaling may be the best approach when you reach the limits of individual EC2 instances.

To confirm that you have the correct EC2 instance size, we recommended that you test your NGINX configuration for your expected and un-expected traffic patterns. The Performance Baselines section of this whitepaper discusses options for testing your configuration. After you are running the desired infrastructure configuration in production, it is also important to continue to monitor the performance of your infrastructure to confirm that it is performing as needed. CloudWatch provides a good option for being able to monitor your infrastructure and confirm that it is performing well. The monitoring section of this whitepaper dives deeper into using CloudWatch for monitoring performance of your NGINX applications within AWS.

A great benefit of running on AWS is that you can start instances, test them for performance, resize to a new instance if you determine that the instance size is not correct for your use, and then terminate the original instances. Depending on the options you chose, you would not have to continue paying for an instance after it is terminated. This approach allows you to try out a variety of instances, so you can find the right fit, at a low initial investment cost.

After you identify the correct instance size, you may still have a need to scale up resources to support your application or service. One of the benefits of running solutions on AWS is the ability to scale up or down based on demand for your application or service. This is done through Auto Scaling and allows the ability to add additional compute resources horizontally to support your workloads instead of vertically scaling to larger instances. With Auto Scaling, you rest assured that while you may have tested a certain load on a particular instance, you have the ability to grow your infrastructure easily to handle additional load.

Application Configuration

In general, it is better to let NGINX Plus decide on the default and auto values of the parameters to various configuration directives controlling worker process behavior and the mechanisms used to interact with the network subsystem. Modern Linux-based operating systems already provide enough intelligence to NGINX Plus in regards to auto-configuration.

Worker Processes

Within NGINX, worker processes do the actual processing of requests received by the NGINX server. We recommend letting NGINX Plus decide on the number of worker processes by setting *worker_processes* to *auto.* The NGINX server is aware of the system resources available to it and the *auto* setting allows NGINX to run one worker process per core, which generally gives the best performance.

If you are running other CPU-intensive services on the same EC2 instance as NGINX, the *worker_processes* value may need to be set to a lower value to limit the amount of CPU that NGINX can use.

If you are running NGINX with a very large disk-based cache, the *worker_processes* value may need to be increased. If your cache is larger than your available memory, then NGINX needs to read from disk; if these disk reads become frequent, it may be necessary for NGINX to consume additional CPU resources.

Concurrent User Connections

If your implementation is creating a large number of concurrent user connections to backend application servers, it is important to verify that there are enough local port numbers available for outbound connections to the backend application.

Verification of the server port range can be done using the following command:

```
$ sysctl net.ipv4.ip_local_port_range
```

If the range needs to be increased, that can be done using the following command:

```
$ sudo sysctl -w net.ipv4.ip_local_port_range="1024
64000"
```

TCP Throughput

As usage of your NGINX instance increases, it is possible that the number of connections and the number of requests to your instance is too great and you begin to experience packet loss, unsuccessfully processed requests, or long latency in request processing times because the kernel cannot keep up with the request rate. One way to overcome additional load on your instances is by implementing Auto Scaling to add additional infrastructure to handle the additional inbound requests.

In addition to Auto Scaling, it may also be necessary for you to investigate kernel settings of your instance to ensure that it is configured to correctly to handle the amount of traffic that your NGINX instance is receiving. This level of troubleshooting and tuning is out of the scope of this whitepaper. Addressing this could involve multiple configurations within your instance. Additionally, it is important to verify if your NGINX configuration also needs adjustment so that it is able to complement any kernel settings that have been changed.

Performance Baselines

To confirm that your NGINX instances are performing optimally for your anticipated workloads, it is important to baseline key aspects of your instances.

There are many performance measurement tools available which are designed to allow you to execute large volumes of requests against your NGINX instance and get an idea on how it performs under load. We recommend that you use not only these tools and the metrics that they provide but also the CloudWatch metrics that are related to the particular EC2 instances that you are using to host NGINX. Looking at both of these together gives you a more complete picture of how your NGINX configuration is performing.

A few generally available tools for executing basic load tests are listed below. Each of these tools gives you the ability to execute basic HTTP queries against your NGINX instance, in addition to a few other features. Additionally, each of these tools provides basic reporting on performance metrics related to the test. There may be deeper pieces of functionality that one tool offers over the other, which may best suit your testing needs. This list is intended to be a guide on possible testing tools and there are many

more tools available than are listed here. In the end, it's up to you to choose the tools that best meet your needs.

- ab (Apache Bench)
- jmeter
- http_load
- siege
- curl-loader
- weighttp
- httperf
- wrk
- sslswamp

When you execute tests, we recommend that the test tool be run on a different instance from the instances that are hosting NGINX so that there is no contention for resources during the testing phase. Additionally, the instances that are executing the tests should be in different Availability Zones and potentially different regions, to accurately simulate realistic traffic conditions, especially if testing against Elastic Load Balancing.

It is important to note that the tools listed above provide basic tests of your NGINX instances and that it may be necessary to also execute more detailed tests related to specific functionality that you are trying to accomplish with your NGINX install.

Load Balancing

To balance traffic for services or applications, NGINX can be implemented as a stand-alone load balancer or integrated behind Elastic Load Balancing. For either of these options, there are architecture and configuration options that need to be taken into consideration.

Integrating NGINX Plus with Elastic Load Balancing

Within AWS, it is common to place NGINX instances behind a load balancer. Elastic Load Balancing can handle HTTP and HTTPS traffic or pass any TCP connection to NGINX instances, which could be performing a variety of functions (web server, reverse proxy, cache server, and so on).

After your NGINX instances are deployed and configured to take traffic, you can deploy an Elastic Load Balancing configuration to serve as a central point for routing traffic, in a balanced method, to your NGINX instances.

Figure 1 outlines an AWS deployment using Elastic Load Balancing and NGINX Plus for additional traffic routing.

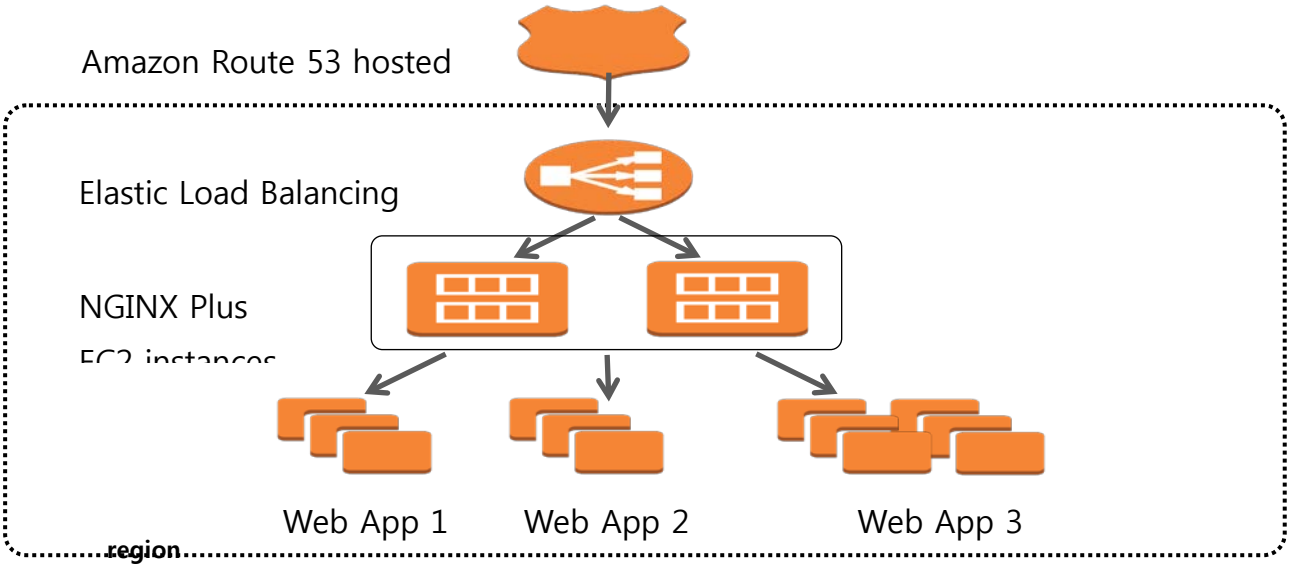


Figure 1: Elastic Load Balancing with NGINX

To most accurately set up NGINX Plus for proxying WebSocket behind a load balancer, support for the *PROXY protocol* should be activated in the NGINX configuration. The NGINX configuration snippet below outlines an example of how to implement support for the PROXY protocol:

```
http {
...
log_format elb_log '$proxy_protocol_addr - $remote_user
[$time_local] ' '"$request" $status
$body_bytes_sent "$http_referer" '
'"$http_user_agent"';

    server {
        listen 80 proxy_protocol;
        set_real_ip_from 172.31.0.0/20;
        real_ip_header proxy_protocol;
        access_log /var/log/nginx/elb-access.log
elb_log;
        location / {
            try_files $uri $uri/ /index.html;
        }
    }
}
```

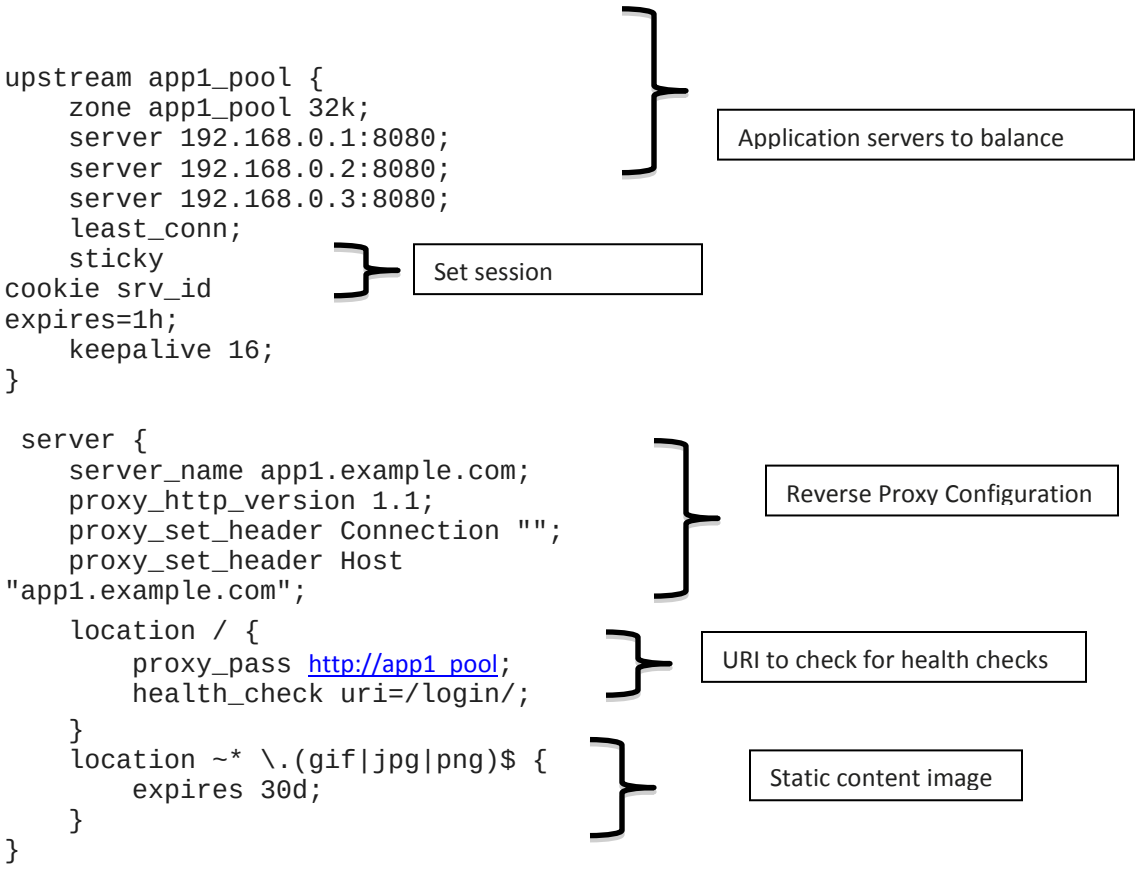
Add PROXY protocol to config and define CIDR

NGINX as a Standalone Reverse Proxy and Load Balancer

One of the core applications for NGINX Plus is as a lightweight, software-only, reverse proxy with load balancing.

The following configuration snippet shows how to load balance requests between three application servers, using HTTP/1.1 with keepalive connections.

With this configuration, requests are distributed based on the number of active connections between NGINX Plus and application servers. This configuration also outlines storage locations for static content. Additionally, with this configuration, proactive health checks poll the */login/* URI every 5 seconds, ensuring continuous application health monitoring.



When you implement an NGINX load balancer solution on Amazon EC2, we recommend that you allocate and associate an Elastic IP address to the EC2 instance that is serving as your load balancer. Allocating and assigning an Elastic IP address ensures that you have a consistent IP for connections into your load balancer and gives you the flexibility to move your load balancer to other EC2 instances while still being able to maintain the same IP address for the entry point into your load balancer.

If your NGINX load balancer is routing traffic to EC2 instances that are part of an Auto Scaling group, it is important to note that the NGINX server needs to be made aware of any additional EC2 instances that are added to your Auto Scaling group. This update to the NGINX configuration file needs to be made manually or through a script that is run after additional instances are added or removed through Auto Scaling. When you use the AWS CLI, the `describe-auto-scaling-groups` command can be used to get a current listing of all EC2 instances that are currently launched into a designated Auto Scaling group. This list of instances can be used to determine if additional instances need to be included in the NGINX load-balancing configuration. For more information about `describe-auto-scaling-groups`, see the AWS CLI Reference. To obtain

the IP address of EC2 instances, listed in the output of the `describe-auto-scaling-groups` command, the `describe-instances` command can be run for each identified instance. For more information about `describe-instances`, see the AWS CLI Reference.

For more information about the request routing, reverse proxy, and load balancing capabilities of NGINX Plus, see the following pages on the NGINX.com site:

- Load Balancing with NGINX and NGINX Plus
- Load Balancing with NGINX and NGINX Plus part 2
- NGINX and NGINX Plus Admin Guide

Security

In addition to ensuring that your EC2 instances are performing correctly and are configured correctly, it is equally important that your EC2 instances are secured properly. Implementing security controls on EC2 instances running NGINX ensures that access is only granted to people or resources that are authorized to access the instances.

Securing NGINX Plus and EC2 instances

For remote user access to NGINX instances running on Amazon EC2, we recommend that you use secure and encrypted communication protocols. The most common use case for this is Secure Shell (SSH). We also recommend that you use public-key authentication over SSH instead of a password, when you control access to your EC2 instances. Using key/cert-based authorization through SSH reduces the exposure of your EC2 instances against password attacks such as dictionary and brute-force attacks.

For each user needing remote access, generate an SSH key pair and keep the private portion of the key pair on the appropriate host for the user needing the access. The public portion of the key pair should be placed in the .ssh directory of the home directory of the user, on the instance that remote access is being granted.

For more information about additional tips and guidance on securing your EC2 instance, see the "Tips for Securing your EC2 Instance" article (http://aws.amazon.com/articles).

Security Groups for EC2 Instances running NGINX

Based on the NGINX recommendations for software configuration, there are specific security group configurations that you need to make. The configuration options in the following table cover inbound rules needed for default NGINX configurations.

Connection Method	Protocol	Port Range	Source IP or Group	Comments
HTTP	tcp	80-80	CIDR IP range that is allowed to access your instances	Port to allow non-encrypted web traffic

HTTPS	tcp	443-443	CIDR IP range that is allowed to access your instances	Port to allow encrypted web traffic
SSH	tcp	22-22	CIDR IP range that is allowed to access your instances	Port to allow SSH access to the instance
SSH	tcp	873-873	CIDR IP range that is allowed to access your instances	Port to allow rsync access to the instance
SSH	udp	5405-5405	CIDR IP range that is allowed to access your instances	Allows Corosync traffic to the instance. Used for high availability configuration.

As you configure your EC2 instances running NGINX, you may implement configurations that use different ports and require network traffic to flow to those ports. Each additional port configuration that is implemented needs to be added to the security group configuration attached to the EC2 instances supporting your NGINX installation.

For each class of NGINX instances that are implemented (reverse proxies vs. web servers for example), we recommend that a separate security group be created for each class. This allows the greatest flexibility in controlling access to your NGINX hosts without being impacted by a configuration that is also used for other non-NGINX instances.

App-Level Security

NGINX Plus includes security-related features that can be used to mitigate erratic or unwanted behavior from a client. Through configuration in NGINX Plus, you can limit connections and requests as well as define access control lists (ACL) for different locations/URI paths.

The configuration snippet below outlines a configuration for enforcing application level security:

```
limit_req_zone $binary_remote_addr
zone=one:10m rate=10r/s;
limit_conn_zone $binary_remote_addr
zone=addr:10m;
server {
    server_name app1.example.com;
    location / {
        limit_req zone=one burst=30;
        proxy_pass http://app1_pool;
    }
    location /downloads/ {
        root /data/downloads;
```

Sets a limit on the requests per second

Limits the number of connections, per IP, for downloads

```
        limit_conn addr 2;
    }
    location /private/ {
        root /data/private;
        allow 127.0.0.1;
        deny all;
    }
}
```

Only allow access to this IP address

For more information about how you can enforce application security with NGINX, see the Restricting Access topic on the NGINX website.

Architecting for High Availability

Common usages of NGINX involve supporting applications and services that must be up and running all the time and ready to support a large number of requests. With this in mind, it is important that you keep high availability architecture in mind when deploying your NGINX instances within AWS. Some key high availability items to keep in mind are:

- Placing NGINX instances in multiple Availability Zones, within a region. This approach allows for the ability to handle failures within or related to a particular Availability Zone.
- Using Auto Scaling to confirm that a minimum number of healthy instances are available to support your application or services. Auto Scaling also provides the benefit of being able to add additional instances if the load on your application or service requires additional resources.

NGINX High Availability Configuration on AWS

In addition to the high availability (HA) architecture options that AWS offers, NGINX also includes configuration that creates an HA cluster of NGINX instances running on Amazon EC2.

It is often desirable to have an IP address which is highly available, regardless of infrastructure failures or changes behind the IP address. With this approach, a single IP is mapped to a particular EC2 instance. When that instance fails or the Availability Zone that the instance is in becomes un-reachable the IP address is mapped to an EC2 instance that is healthy or in another Availability Zone. This type of HA configuration allows for the appropriate resources to be available to handle requests for your application even in the event of an unexpected failure of an instance. NGINX Plus offers the ability to have this type of configuration on AWS.

The NGINX HA solution provides the following:

1. Configuration allowing for two instances running NGINX Plus to serve your application needs.
2. Corosync and Pacemaker configuration to monitor the health of your instances.
3. Allocation of an Elastic IP address to the primary NGINX node.
4. Re-allocation of the Elastic IP address to the secondary NGINX instance when the primary is unhealthy or unreachable.

A core piece of the NGINX HA configuration includes use of Pacemaker and Corosync, which are clustering solutions for Linux. Corosync offers a clustering engine that facilitates the ability to create HA for applications and facilitate communications among the instances in a cluster. Pacemaker takes advantage of the messaging capabilities provided by Corosync and monitors the instances in the cluster for hardware and software failures. If a failure is detected, Pacemaker recovers on healthy instances in the cluster.

The following configuration steps outline how to set up a cluster of NGINX servers running as an active/passive pair with one shared IP address.

Pre-Configuration Steps

Before you configure your NGINX HA instances, there are several prerequisites:

1. You must have at least one Elastic IP address available in the region in which you are creating the HA configuration. For an Elastic IP address, available means that the IP address has been allocated but has not been assigned to an EC2 instance.
2. You must create an IAM role and an IAM instance profile, which grants the NGINX HA configuration the appropriate permissions on your EC2 instances. The instance profile must then be assigned to the EC2 instances running NGINX that will be used for HA configuration. The instance profile can only be assigned when EC2 instances are first launched and cannot be assigned to instances that are already launched.
3. If you have existing EC2 instances running NGINX Plus and intend to deploy this HA solution on those instances, you first need to launch new instances with the IAM instance profile created in step 2.

Allocating an Elastic IP Address

Use the following steps to create a new Elastic IP address.

- In the AWS Management Console, choose the Amazon EC2 service.
- Under the Network and Security section, choose Elastic IPs.
- Click **Allocate New Address** and then choose the option to acknowledge that you want to create a new Elastic IP address.

Alternately, you can allocate an Elastic IP address with the AWS CLI. The following CLI command allocates a new Elastic IP address:

```
syntax:
$ aws ec2 allocate-address –domain vpc [--region
<region_name>]
```

```
usage:
$ aws ec2 allocate-address --domain vpc --region us-
east-1

output:
{
```

```
            "PublicIp": "54.86.222.93",
            "Domain": "vpc",
            "AllocationId": "eipalloc-1e4e827b"
        }
```

Defining an Instance Role for Use with NGINX HA Configuration

Instance roles provide the ability to manage credentials for applications running on EC2 instances. Instance roles define the permissions that an application would need and grants the necessary permissions for that application to call AWS services. For more information about instance roles, see Granting Applications that Run on Amazon EC2 Instances Access to AWS Resources in the *Using IAM Guide*.

To meet the needs of the NGINX HA functionality, create a role, which allows the following permissions:

- Ability to assign an Elastic IP address with an instance
- Ability to disassociate an Elastic IP address with an instance
- Read-only permissions to run Amazon EC2 `describe*` commands

This section outlines how to create the necessary instance role manually using the IAM console or using the AWS CLI.

To configure an instance role manually

1. In the IAM console, define the overall role, which is used to manage the policies for the NGINX HA instances:

    - In the IAM Resources section, click **Roles**.
    - Click **Create Role**.
    - Enter a role name, and click **Next Step**.
    - In the Select Role Type section, choose Amazon EC2 for the service role and click **Select**.
    - In the Set Permissions section, choose Custom Policy and click **Select**.
    - Enter the policy name and the policy document in the Set Permissions form (Figure 2).

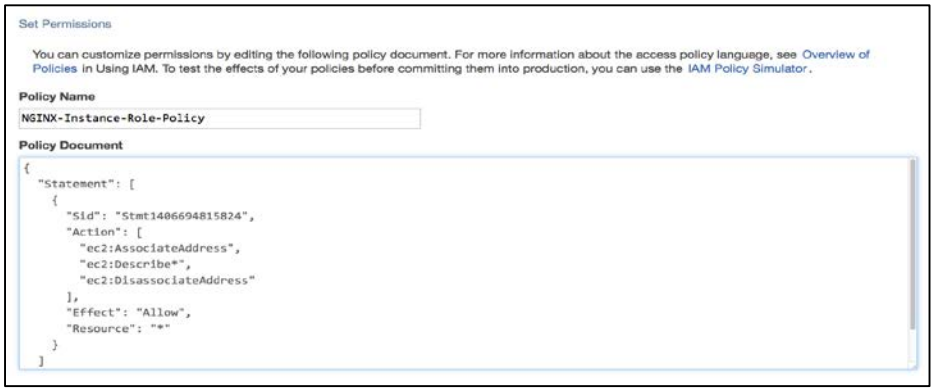


Figure 2: Custom policy for the role

** You can copy and paste the above policy document from the next section of this whitepaper, configuring an instance role using the AWS CLI.

To configure an instance role using the AWS CLI

1. Define the overall role, used to manage the policies for the NGINX HA instances.

- On the instance where you will be running the AWS CLI commands, create a file that contains the following trust policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": { "Service": "ec2.amazonaws.com"},
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Use the following AWS CLI command to create the role and attach the trust policy:

```
syntax:
$ aws iam create-role –role-name <role-name> --assume-role-
policy-document file://<file-location>
```

```
usage:
$ aws iam create-role --role-name NGINX-Instance-Role --assume-
role-policy-document file:///tmp/nginx_trust_policy.txt

output:
{
    "Role": {
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Action": "sts:AssumeRole",
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "ec2.amazonaws.com"
                            }
                    }
                ]
            },
            "RoleId": "AROAJMNJKBBBX4B7ACUUI",
            "CreateDate": "2014-07-29T23:15:42.978Z",
```

```
            "RoleName": "NGINX-Instance-Role",
            "Path": "/",
            "Arn": "arn:aws:iam::526039161745:role/NGINX-
Instance-Role"
        }
}
```

2. Assign the appropriate policies to the new role.

On the instance where you will be running the AWS CLI commands, create a file that contains the following policy:

```
{
  "Statement": [
    {
      "Sid": "Stmt1406694815824",
      "Action": [
        "ec2:AssociateAddress",
        "ec2:Describe*",
        "ec2:DisassociateAddress"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

- Use the following AWS CLI command to associate the policy with the role:

syntax:

$ aws iam put-role-policy --role-name <role name defined above> --policy-name <policy name from above table> --policy-document file://<location of policy document>

```
usage:
$ aws iam put-role-policy --role-name NGINX-Instance-
Role --policy-name NGINX-Instance-Role-Policy --policy-
document file:///tmp/ nginx-instance-role-policy.txt

output:
There is no output for a successful execution of this
command
```

When you create an instance role with the AWS CLI, you need to take additional steps to create the instance profile and to associate the role with the instance profile. These are steps that occur automatically when creating a role in the IAM console.

- Use the following AWS CLI command to create the instance profile:

```
syntax:
$ aws iam create-instance-profile --instance-profile-
name <instance profile name>
```

```
usage:
$ aws iam create-instance-profile --instance-profile-
name NGINX-Instance-Role

output:
{
    "InstanceProfile": {
        "InstanceProfileId": "AIPAJFTAWANM36EU4DWAU",
        "Roles": [],
        "CreateDate": "2014-07-30T03:58:58.188Z",
        "InstanceProfileName": "NGINX-Instance-Role",
        "Path": "/",
        "Arn": "arn:aws:iam::526039161745:instance-
profile/NGINX-Instance-Role"
    }
}
```

Use the following AWS CLI command to associate the instance profile with the role:

```
syntax:
$ aws iam add-role-to-instance-profile –instance-
profile-name <instance profile name> --role-name <role
name>
```

```
usage:
$ aws iam add-role-to-instance-profile --instance-
profile-name NGINX-Instance-Role --role-name NGINX-
Instance-Role

output:
There is no output for a successful execution of this
command
```

At this point, all setup is done for the instance role and you can move onto assigning the instance role to your EC2 instances.

Assigning Your Instance Role to EC2 Instances

The instance role is now ready to be attached to newly launched EC2 instances, which will be used for NGINX HA.

If you launch EC2 instances via the Amazon EC2 console, the instance role can be attached to your NGINX instances by selecting the role in the Configure Instance Details section of the EC2 Launch Instance wizard (Figure 3).

Additionally, two EC2 instances, in the same region, are needed to support the NGINX HA configuration. Ideally, these instances will be located in different Availability Zones in line with recommended best practices for high availability on AWS. When launching an instance in the Amazon EC2 console, the Availability Zone can be set in the Subnet section of the EC2 Launch Instance wizard (Figure3).

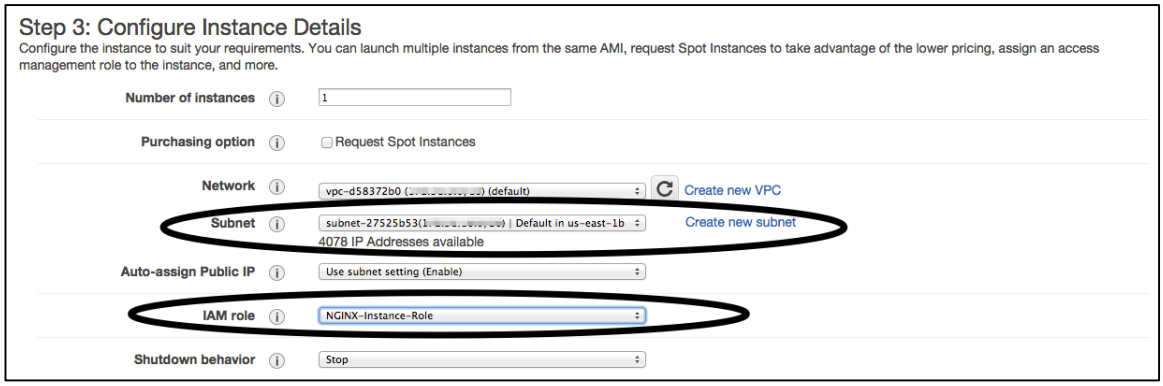


Figure 3: Configure instance details

EC2 instances can also be launched using the AWS CLI. The relevant command for launching instances is `aws ec2 run-instances`. The `run-instances` command has many parameters that can be used to launch an EC2 instance. Which parameters you choose depend on the configuration you are trying to run for your EC2 instances. For launching two instances with the appropriate IAM role, the following parameters are relevant:

- count: Parameter for defining the number of instances to launch.
- iam-instance-profile: Name of the instance profile to attach to the EC2 instances.
- subnet-id: ID of the subnet to launch the instance in. Used for multi-Availability Zone instance configuration

HA Configuration Steps

Now that the pre-configuration steps are complete, the process of actually configuring NGINX on the EC2 instances can be started.

The first step in setting up an HA pair of NGINX Plus servers is to install the *nginx-ha* package on both EC2 instances.

On NGINX Plus Amazon Linux AMI, install *nginx-ha* with the following command:

```
$ sudo yum install nginx-ha
```

On NGINX Plus Ubuntu AMI, the install command is:

```
$ sudo apt-get install nginx-ha
```

After successful installation, run the setup command:

```
$ sudo nginx-ha-setup
```

The *nginx-ha-setup* script should be invoked on both EC2 instances, simultaneously. The script on each instance then prompts for:

- The EC2 instance ID of the other HA peer instance
- The host name
- The Elastic IP address to be made highly available

Throughout the configuration process, you need to switch back and forth between each instance, as each step of the installation process needs to be run in sequence on each of the instances.

When you complete the configuration, a functioning, two-node active/passive cluster should be up and running and an Elastic IP address is assigned to the instance that you designated as primary during the configuration process. The status of the running cluster can be determined by running the following command from either of the instances that were just configured:

```
$ sudo crm status bynode
```

Here is the sample output of the crm status bynode command for two nodes named *nginxha100* and *nginxha101*, and with the Elastic IP address running on *nginha100:*

```
    ============
    Last updated: Wed Mar 19 02:46:49 2014
    Last change: Wed Mar 19 02:46:42 2014 via cibadmin
on nginxha101
    Stack: openais
    Current DC: nginxha101 – partition with quorum
    Version: 1.1.6-
9971ebba4494012a93c03b40a2c58ec0eb60f50c
    2 Nodes configured, 2 expected votes
    2 Resources configured.
    ============
    Node nginxha100: online
    ha-ip (ocf::heartbeat:IPaddr2) Started
    ha-nginx (ocf::nginx-ha:nginx-ha) Started
    Node nginxha101: online
```

If everything is working correctly, running the following command produces the same output on both nodes:

```
$ sudo crm status bynode
```

Details of the Pacemaker configuration can be obtained by running the following command:

```
$ sudo crm configure show
```

NGINX HA Architecture Considerations

When you implement the NGINX HA configuration, there are several different architecture options that you can choose. A key point to keep in mind for determining the correct architecture is that Elastic IP addresses are allocated at the AWS region level. This means that you cannot have an HA configuration that relies on one Elastic IP address being shared between EC2 instances in different regions.

A basic architecture for NGINX HA would consist of two EC2 instances running the NGINX HA configuration, with each EC2 instance in a different Availability Zone of an AWS region.

Figure 4 outlines a sample architecture, at the AWS region level, using the NGINX HA configuration. There are lots of ways to architect the web apps that sit below the NGINX instances. The point to take note of in this diagram is that the NGINX HA instances are in different Availability Zones within an AWS region. This acts as a safeguard so that if there are problems in an Availability Zone, the failover instance can successfully take over work from the primary.

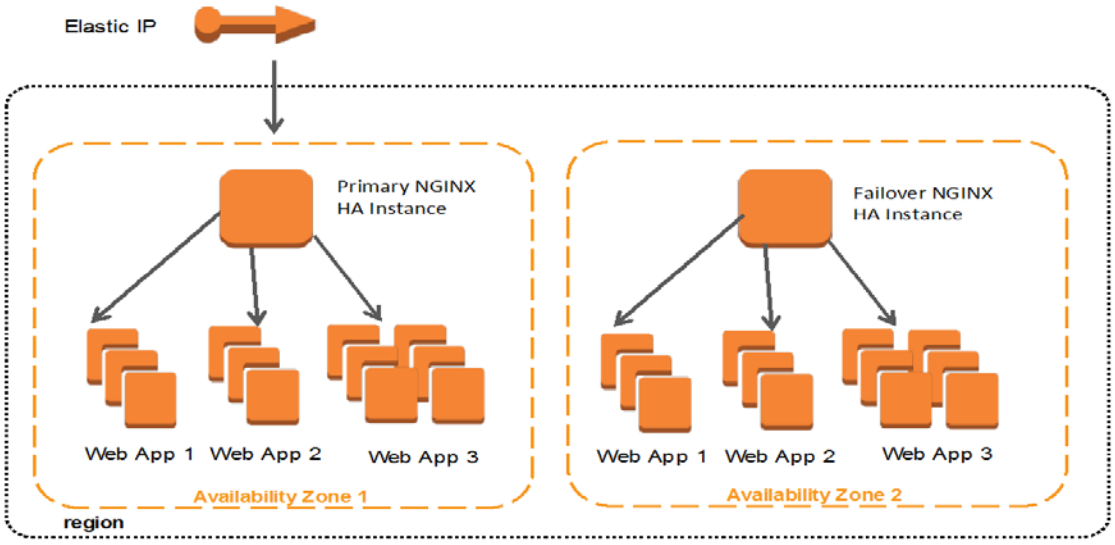


Figure 4: Sample NGINX high availability architecture

Using Amazon Route 53 offers other opportunities to have multiple groups of NGINX HA configurations distributed among multiple AWS regions. This configuration could be used to support a disaster recovery scenario or even to support load balancing across AWS regions. With this approach, an Elastic IP address would be allocated in each region that is intended to support the NGINX HA configuration. The necessary HA configuration would be done on each of the EC2 instances within each region, resulting in the Elastic IP address in each region being mapped to a primary NGINX instance. Amazon

Route 53 can then be configured to support routing policies that best support your needs. These routing policies include Simple, Weighted, Latency, Failover, and Geolocation. For more information about Amazon Route 53 routing policies, see Choosing a Routing Policy in the *Amazon Route 53 Developer Guide*.

Figure 5 shows an example architecture with two NGINX HA configurations in different regions supported by Amazon Route 53 for request routing.

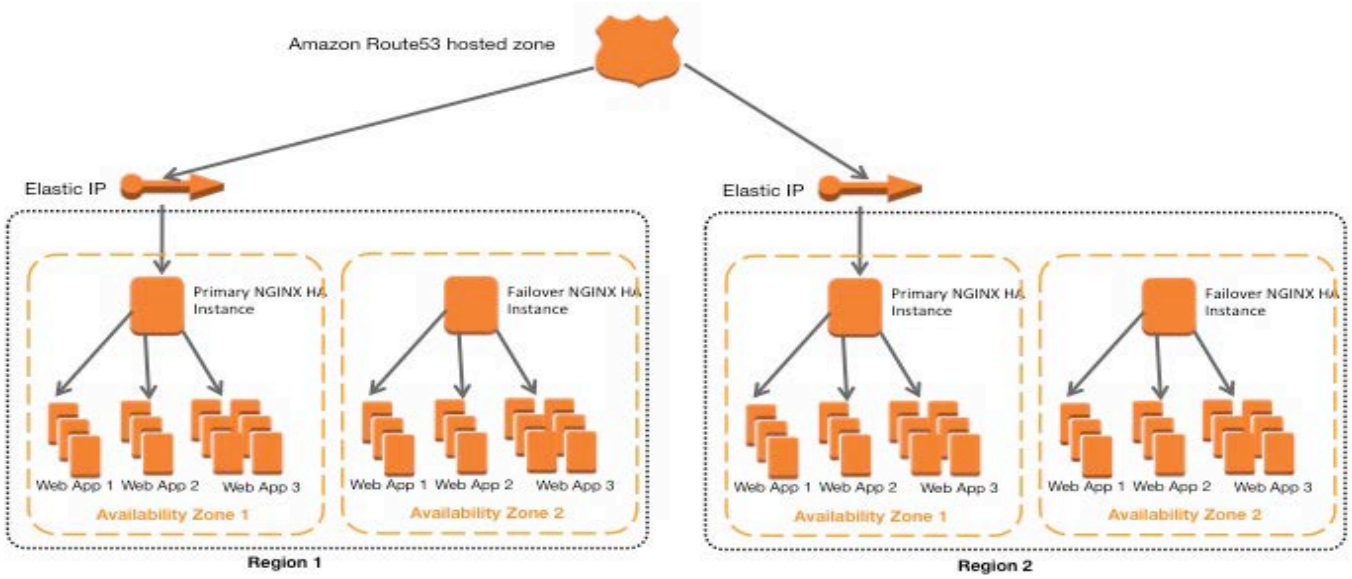


Figure 5: NGINX high availability across regions

Additional Configuration Considerations

After completing installation of the NGINX HA solution on Amazon EC2, it is important to verify that both EC2 instances have the appropriate configuration for the services that are being made highly available. Additionally, as changes are deployed to these instances, it is important that they are deployed to both instances to ensure that application functionality is unchanged in the case of a failover from one instance to another.

The default timeout used by Corosync to consider a node as down is 1 second. This default timeout, as well as the other parameters can be changed in Corosync configuration (check */etc/nginx-ha/templates/corosync.conf.tmpl*).

The installation and configuration of NGINX for high availability comes with the default configuration for Corosync and Pacemaker. This allows for an active/passive configuration with two NGINX instances. If needed, more complex configurations such as active/active or using more than two instances can be created. Making these changes requires changes to the Pacemaker configuration. For more information about configuration changes to Pacemaker, see the ClusterLabs documentation.

Monitoring

Monitoring what is going on with your applications and EC2 instances is just as important as the initial configuration and performance testing. Through proper monitoring, you can have visibility into how your application and infrastructure are performing under various real-world conditions. Additionally, proper monitoring provides you with the opportunity to be able to take action at key moments before they become problems, keeping risk to your application or service low.

Running NGINX on EC2 instances provides several different opportunities to monitor what is going on with your instances and be positioned to take action before your application or users are negatively impacted.

Amazon CloudWatch

With CloudWatch, you can have a complete monitoring solution for all aspects of your NGINX instance and the applications that it is serving. When you run NGINX on EC2 instances, you get access to a group of metrics that are delivered for all EC2 instances. By default, the following metrics are presented for all EC2 instances:

- CPU Utilization (Percent)
- Disk Reads (Bytes)
- Disk Read Operations (Operations)
- Disk Writes (Bytes)
- Disk Write Operations (Operations)
- Network In (Bytes)
- Network Out (Bytes)
- Status Check Failed (Count)

For more information about each of the default EC2 metrics, see Amazon Elastic Compute Cloud Dimensions and Metrics in the *Amazon CloudWatch Developer Guide*.

Integrating NGINX Metrics with CloudWatch

CloudWatch supports the ability to import custom metrics related to your NGINX instances running on Amazon EC2. If you integrate these custom metrics into CloudWatch, you can view these metrics on their own or along with other Amazon EC2 or custom metrics. Additionally, custom metrics can be used to build alarms against your application metrics.

NGINX provides a set of metrics that give insight into activity related to work that NGINX is doing. Additionally, NGINX also provides functionality, which gives the ability to import these metrics into CloudWatch. The following section outlines how to configure delivery of NGINX metrics to CloudWatch.

Configuring NGINX to Deliver Metrics to CloudWatch

Configuring NGINX to deliver metrics to CloudWatch is a multi-step process. This process involves:

- Configuring NGINX to collect and report its metrics

- Granting permission for your EC2 instance to post metrics to CloudWatch
- Installing and configuring the package for sending NGINX metrics to CloudWatch
- Running background process to collect and deliver metrics to CloudWatch

To configure NGINX to collect and report its metrics

For more information about how to enable the specialized monitoring for NGINX, see the Live Activity Monitoring topic on the NGINX.com site.

To grant permission for your EC2 instance to post metrics to CloudWatch

Permission for posting metrics from an EC2 instance to CloudWatch is controlled through IAM instance roles. To enable delivery of NGINX metrics to CloudWatch, the instance role tied to the EC2 instance on which NGINX is running needs to be updated, or a new instance role needs to be created and assigned to the EC2 instance.

If you already have an instance role tied to the EC2 instance on which NGINX is running, then the role needs to be updated with an additional policy. A standalone policy to allow CloudWatch access looks like the following:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1398821442000",
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

If a new instance role needs to be created to support CloudWatch integration or if the CloudWatch policy needs to be added to an existing instance role, the details of how to accomplish this are outlined in the manual instance role configuration section of this whitepaper.

To install and configure the package for sending NGINX metrics to CloudWatch

Integration with CloudWatch is accomplished with the nginx-cw-agent package.

On the NGINX Plus Amazon Linux AMI, install nginx-cw-agent with the following command:

```
$ sudo yum install nginx-cw-agent
```

On NGINX Plus Ubuntu AMI, the install command is:

```
$ sudo apt-get install nginx-cw-agent
```

After the package has been installed, the next step is to update the agent configuration file with details about where the status information can be retrieved. The configuration file is located at: /etc/nginx-cw-agent/nginx-cw-agent.ini

At the bottom of the nginx-cw-agent.ini file, are examples of how to structure the configuration for retrieving status information. The two possible configurations are:

```
[source1]
name=exampleorg
url=http://example.org/status

[source2]
name=examplecom
url=http://example.com/status
http_user=testuser
http_pass=testpass
```

For each of the examples, the names configuration represents the name that is given to the group of metrics when they are viewed in CloudWatch. If you use a meaningful value for the namespace, that helps you find the metrics easily in CloudWatch. The URL configuration represents the URL path to the NGINX host from which the status is collected. The http_user and http_pass values represent a username and password for accessing the status page, if authorization for accessing your application is enabled.

In addition to the source configuration, the global configuration of pool_interval can also be changed. This configuration represents the frequency, in seconds that the status metrics is polled for delivery to CloudWatch.

To verify metrics background processes

After you complete the configuration, you can submit the agent to collect metrics and send them to CloudWatch. Before you submit the background agent, you can test your configuration and verify that metrics are being collected. Running the following command enables the agent to run in the foreground and give visibility as to whether metrics are successfully being collected and posted:

```
$ /usr/bin/nginx-cw-agent.py -f start
```

After you have verified the metrics agent configuration file, you can submit the agent to run in the background. The following command enables the agent to run in the background:

```
$ sudo service nginx-cw-agent start
```

The logging for the agent is written to the /var/log/nginx-cw-agent.log file for any analysis or troubleshooting needs.

After the metrics agent is running, the collected metrics are available in CloudWatch. To view your metrics, navigate to the CloudWatch dashboard in the AWS Management Console.

On the dashboard, click **Browse Metrics.**

On the "CloudWatch Metrics by Category" page click the Custom Metrics list and select the value that you chose for the name parameter of the NGINX metrics agent. Choosing this value takes you to the CloudWatch metrics page related to the metrics for your NGINX metrics.

In addition to the custom NGINX metrics discussed above, CloudWatch supports the ability to import any other custom metrics that you define. For more information about how to import additional custom metrics into CloudWatch, see Publish Custom Metrics in the *Amazon CloudWatch Developer Guide*.

Integrating NGINX Logs with CloudWatch

CloudWatch enables you to import and store your application log files. After the log files are imported, CloudWatch can serve as your log file repository so you can retain log files for a period of time that you define and then rotate out log files that are older than the retention period. Additionally, CloudWatch allows you to scan your log files for particular patterns and then turn these into metrics that can be viewed and reported on within CloudWatch. After metrics are defined for the contents of your log files, you can also create alarms against the metrics and send notifications or take actions such as Auto Scaling.

NGINX provides the ability to configure error and access logging. Error logs provide insight into errors of various severity levels, which have occurred within the NGINX application. Access logs provide information about client requests after the requests have been processed. For more information about how to configure these logs, see Logging and Monitoring on the NGINX web site.

Both of these types of log files can be imported into CloudWatch for log analysis and metric generation. For example, CloudWatch could be used to count the number of 404 errors within a log file, which would then be an additional metric available for analysis and alarm generation. For more information about CloudWatch logs, including how to integrate your logs into CloudWatch, see Monitoring System, Application, and Custom Log Files in the *Amazon CloudWatch Developer Guide*.

CloudWatch Logs can be used to ingest log files for one to many hosts. If you have multiple hosts running NGINX and you need to monitor the same log file from all hosts, you can consolidate these under one CloudWatch Log group. Within CloudWatch, you have the ability to drill down from a log group down to an EC2 instance and then to the log file entries for that instance. When you create metrics against a log group that covers multiple instances, the metric is at the log group level and thus covers all hosts. Additionally, any alarms created for metrics generated from logs are also at the log group level, allowing you to have alarming at a level covering your overall application fleet.

Figure 6 shows a metric filter, which is configured to capture metrics that match a pattern in the NGINX access.log file. In this case, any 400-type status codes are captured in the metric. When you define a metric filter, you have full flexibility about what information you want to capture from your log files and you can have multiple metric filters that cover one log group.

Figure 6: NGINX access log filter

With the metric filter defined, there is now a custom CloudWatch metric that charts the status codes that the filter pattern is looking for. With a metric in place, you can view the metric data in CloudWatch or create an alarm to take action based on certain thresholds.

Figure 7 shows an example alarm that was created to alarm on a certain amount of 400-type status codes within a designated amount of time.

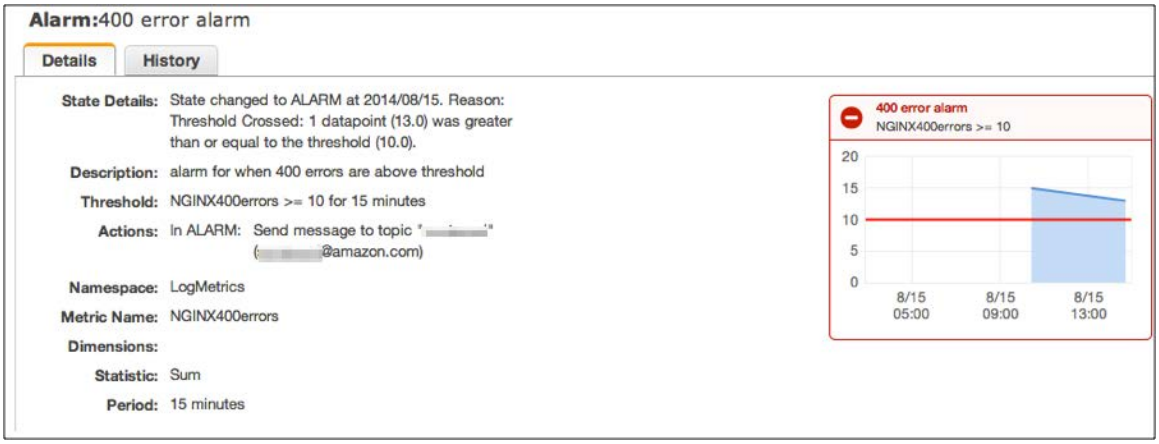


Figure 7: Alarm configuration for 400 type errors

In addition to viewing log messages and metrics within CloudWatch, you can also use the CloudWatch Log API to extract log messages and their metrics from CloudWatch and analyze or import the log information into other applications.

Backup Strategy

It is important to ensure that your NGINX configuration is preserved and able to withstand instance and storage failures. This preservation allows you to recover quickly and completely.

In AWS, the NGINX application code and configuration files are stored on the root volume of the EC2 instance. This root volume is backed by Amazon EBS. With this in

mind, there are several strategies that you can employ to confirm that your current configuration is appropriate. Some of these options are:

1) Create an AMI of your current NGINX production configuration every time that a change is made. Creating an AMI of your current production configuration allows you have a starting point for new or replacement EC2 instances for your NGINX needs. This new AMI should be used for new production instances as well as creating additional instances through Auto Scaling or replacing failed instances. For more information about how to create an AMI from a running EC2 instance, see Creating an Amazon EBS-Backed Linux AMI in the *Amazon EC2 User Guide for Linux*.
2) Create snapshots of the root Amazon EBS volume on the EC2 instance that has your golden production NGINX configuration. This snapshot will allow your current configuration to be preserved and allows you to create a new Amazon EBS volume and recover your configuration as needed. Volumes created from snapshots can be attached to existing instances or new instances. For more information about taking Amazon EBS snapshots, see Amazon EBS Snapshots in the *Amazon EC2 User Guide for Linux*.
3) Store your NGINX configuration files in a version-enabled Amazon S3 bucket. Use the Amazon EC2 user data functionality to have a script run at instance launch, which pulls the current version of the configuration files from Amazon S3 and places them in the correct location on your NGINX instance. For more information about how to implement the Amazon EC2 user data functionality, see Launching Instances with User Data in the *Amazon EC2 User Guide for Linux*. Alternately, create a boot script, built into your NGINX AMI, which grabs the latest configuration file from Amazon S3 upon instance boot, and deploys that file to the appropriate location on your NGINX instance.
4) Use continuous integration software, such as Jenkins, to build AMIs containing your NGINX configuration programmatically, and have them ready for production deployment.

Additional NGINX Configuration Options

In addition to the usage and configuration options mentioned in this whitepaper, there are many other configuration options available with NGINX Plus available to you. These include:

- SSL Termination
  http://nginx.com/resources/admin-guide/nginx-ssl-termination/
  http://nginx.org/en/docs/http/ngx_http_ssl_module.html
- App Edge Caching
  http://nginx.org/en/docs/http/ngx_http_proxy_module.html - proxy_cache
  http://nginx.org/en/docs/http/ngx_http_proxy_module.html - proxy_cache_path
- Streaming Media
  http://nginx.com/solutions/fast-scalable-video-delivery/
- Application Caching
  http://nginx.org/en/docs/http/ngx_http_memcached_module.html

Conclusion

AWS provides a unique set of services for running Internet applications. Pairing NGINX Plus with the existing AWS offerings allows customers the ability to create a fully functioning, high performance, highly available, and secure Internet application. Using NGINX Plus with AWS services such as Amazon EC2, Elastic Load Balancing, Auto Scaling, and CloudWatch allows customers to leverage AWS services while taking advantage of key NGINX functionality that they want to run their applications.

Additional Getting Started Resources

For NGINX documentation, see http://nginx.org/en/docs/.

For the *NGINX Beginners Guide*, see http://nginx.org/en/docs/beginners_guide.html.

For more information about NGINX Plus, see http://nginx.com/products.

For information about AWS, see http://aws.amazon.com.