Setting up your Windows AWS Shell environment

After setting up your security credentials the next thing to do is to set up your command line environment (which we will refer to as your AWS Shell from now on). This shell allows you to interact with the AWS API’s from the command line allowing you to start instances, stop instances, create IAM users etc.

You will need the Java JRE or SDK installed as a pre-requisite.

Create a folder on your local disk and then download and unzip the various AWS command line tools into it. The AWS tools can all be accessed from : <http://aws.amazon.com/developertools>

The tools you will need are:

Amazon EC2 API tools

AWS CloudFormation command line tools

Elastic Load balancing Tools

Autoscaling API tools

Amazon Cloudwatch API Tools

IAM Command Line Toolkit

Amazon Simple Notification Service Command Line Interface Tool

Amazon EC2 AMI Tools

Amazon RDS Command Line Tools

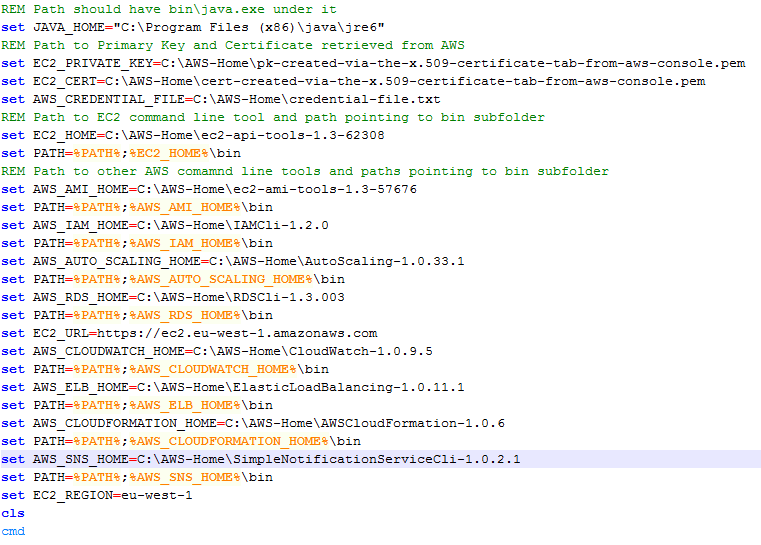
Amazon CloudFront signed URLs helper tools

JavaScript scratchpad for Amazon SimpleDB

If you are going to use the Simple Email Service or Elastic BeanStalk ( a ruby based tool) download those tools as well.

Next create a batch file that will be used to launch your AWS shell with the correct Environment variables and paths.

An example batch file (The example file refers to the latest versions of the command line tools available End of Feb 2011):



There are two ways you can provide your AWS credentials which allow you to actually use the tools via the use of AWS keys, or by using X.509 certificates. These are set via the EC2\_PRIVATE\_KEY and EC2\_CERT if using the x.509 certificate authentication or via the AWS\_CREDENTIAL\_FILE variable. The AWS\_CREDENTIAL\_FILE points to a file that contains the following two lines:

*AWSAccessKeyId=<Write your AWS access ID>*

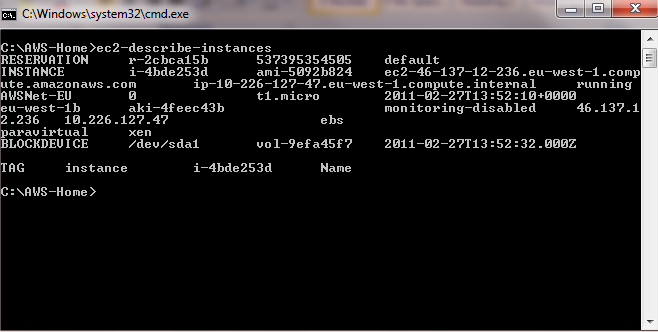
*AWSSecretKey=<Write your AWS secret key>*

Depending on the tool being used determines which authentication method is used. Please make sure you protect the authentication certificates and access keys from unauthorised access.

You will need to set the AWS region you will be working in, in your shell configuration. This is set via the EC2\_REGION and EC2\_URL environment variables. The example file shows you the settings for the eu-west-1 region.

We advise keeping the version number of the Command line tools in the naming of the folders that have been unzipped under your AWS Shell top level folder ( In the example this top level folder is c:\AWS\_HOME). This is so you can keep track of what versions you are using.

Once you have your AWS shell set up start it up and test each tool using one of the commands available. The Screenshot below shows the outcome of using the ec2 tools to find out how many instances are being run by using the ec2-describe-instances command.



Starting from the AWS documentation landing page <http://aws.amazon.com/documentation/> you can access the command line reference and quick reference cards for each of the command line tools. Make use of your free usage tier to try some of the commands out and get used to interacting with AWS from the command line.

Setting up your Windows AWS Shell environment

After setting up your security credentials the next thing to do is to set up your command line environment (which we will refer to as your AWS Shell from now on). This shell allows you to interact with the AWS API’s from the command line allowing you to start instances, stop instances, create IAM users etc.

You will need the Java JRE or SDK installed as a pre-requisite.

Create a folder on your local disk and then download and unzip the various AWS command line tools into it. The AWS tools can all be accessed from : <http://aws.amazon.com/developertools>

The tools you will need are:

Amazon EC2 API tools

AWS CloudFormation command line tools

Elastic Load balancing Tools

Autoscaling API tools

Amazon Cloudwatch API Tools

IAM Command Line Toolkit

Amazon Simple Notification Service Command Line Interface Tool

Amazon EC2 AMI Tools

Amazon RDS Command Line Tools

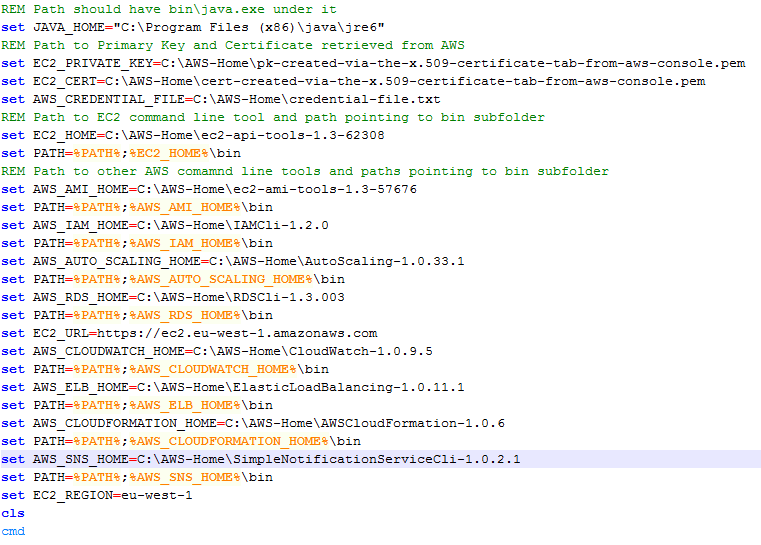
Amazon CloudFront signed URLs helper tools

JavaScript scratchpad for Amazon SimpleDB

If you are going to use the Simple Email Service or Elastic BeanStalk ( a ruby based tool) download those tools as well.

Next create a batch file that will be used to launch your AWS shell with the correct Environment variables and paths.

An example batch file (The example file refers to the latest versions of the command line tools available End of Feb 2011):



There are two ways you can provide your AWS credentials which allow you to actually use the tools via the use of AWS keys, or by using X.509 certificates. These are set via the EC2\_PRIVATE\_KEY and EC2\_CERT if using the x.509 certificate authentication or via the AWS\_CREDENTIAL\_FILE variable. The AWS\_CREDENTIAL\_FILE points to a file that contains the following two lines:

*AWSAccessKeyId=<Write your AWS access ID>*

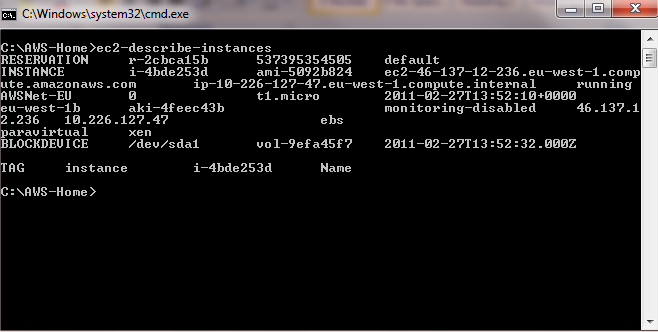
*AWSSecretKey=<Write your AWS secret key>*

Depending on the tool being used determines which authentication method is used. Please make sure you protect the authentication certificates and access keys from unauthorised access.

You will need to set the AWS region you will be working in, in your shell configuration. This is set via the EC2\_REGION and EC2\_URL environment variables. The example file shows you the settings for the eu-west-1 region.

We advise keeping the version number of the Command line tools in the naming of the folders that have been unzipped under your AWS Shell top level folder ( In the example this top level folder is c:\AWS\_HOME). This is so you can keep track of what versions you are using.

Once you have your AWS shell set up start it up and test each tool using one of the commands available. The Screenshot below shows the outcome of using the ec2 tools to find out how many instances are being run by using the ec2-describe-instances command.



Starting from the AWS documentation landing page <http://aws.amazon.com/documentation/> you can access the command line reference and quick reference cards for each of the command line tools. Make use of your free usage tier to try some of the commands out and get used to interacting with AWS from the command line.

Part of moving to the cloud is encouraging the team to embrace the cloud. In our experience the flexibility and ‘power’ it gives can be over whelming and there is often a sense of clinging to the traditional practises so it is imperative to get the team to start accepting new ways of working. One of the ways to help them along his journey is to get the development environment up and running as quickly as possible. This will demonstrate that the platform is no longer a bottle neck as well as providing the proof that the cloud can deliver faster than anything a traditional environment can.

The first environment to be made available is the development environment. Typically this will consist of a set of AMI’s (Amazon machine image) that can be spun up on demand and the provisioning of the ALM toolset (source control, build services and task management).

Working with instances derived from a base AMI is just like working with a base operating system installed on your workstation or a virtual guest on say VMware or Hyper-v.

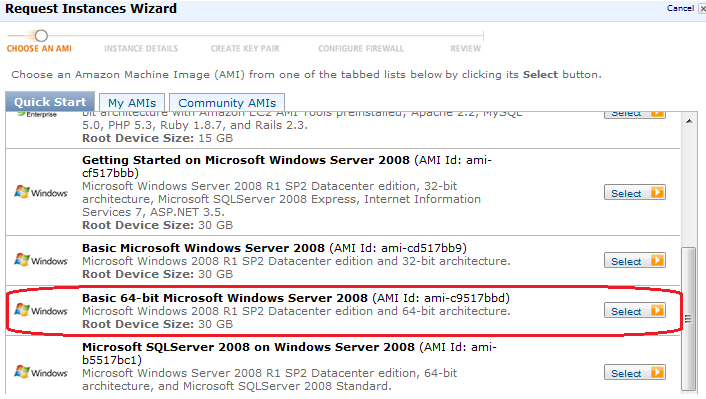
You will need to set up your security group to allow RDP and SSH connections from specified source IPs.

Instantiating and connecting to a Windows instance

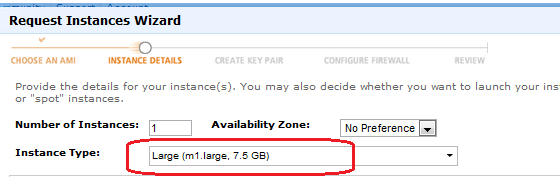
As this is a book about developing .NET applications on AWS it is inevitable that you will be creating some base windows AMIs.

Firstly select the base AMI. We shall select the Amazon windows 2008 x86\_64 AMI (Rumoured at the time of writing that windows 2008 R2 would soon be available)

In our case we are working in the EU region so the AMI we wanted had the AMI id: ami-c9517bbd



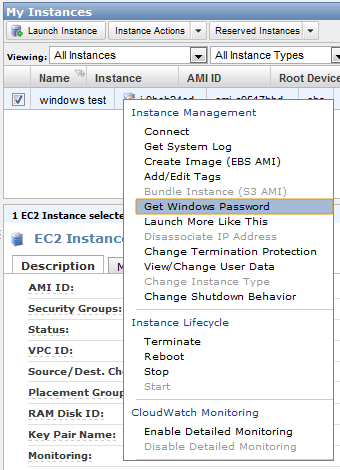
Running a windows instance any smaller than large can prove to be challenging as it is very slow. Using t1.micro is fine once it’s already been configured if you have low throughput web applications or services.



Although you will be using RDP to connect to your windows instance you will need to use your SSH key you created earlier as part of your AWS set up activities.



On first starting a windows instance from an Amazon base AMI you will need to retrieve the administrators password and will need to use your SSH key to achieve this.



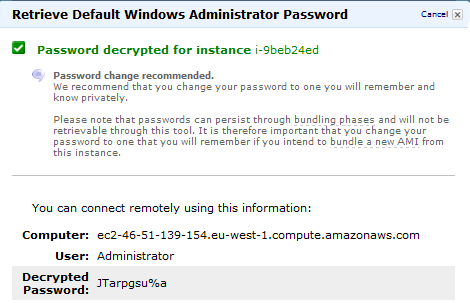
This will be needed for the decryption of the initial Administrators password. Note the warning if you try and retrieve your password before it is ready. This is an ideal time for that coffee break.



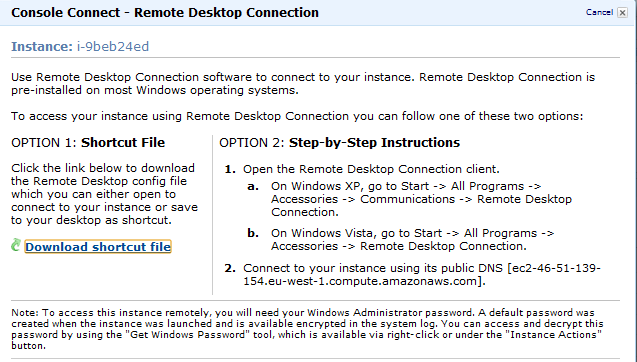
When the instance is ready you will then be prompted to decrypt the password



Use a text editor to open your private key and copy the key into the box click the Decrypt password button and then you will be provided with the randomly generated password.

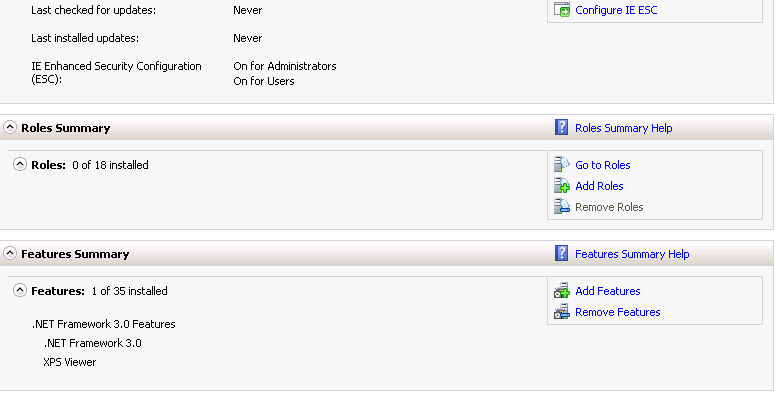


After decrypting you will then want to connect via RDP to your instance and change the administrators password to something you can remember ( Do please use good password selection criteria making sure you use a mix of upper & lower case characters, numbers and non alpha numeric characters and of course keep it reasonably long) . If you select connect from the AWS console you will see the following (They do need to update this information to reflect the increasing use of Windows 7)



Personally I have found connecting via the elasticfox Firefox plug in makes life simpler as you do not have to do the cutting and pasting of the public DNS value but can connect directly to the instance via RDP from the elasticfox plugin

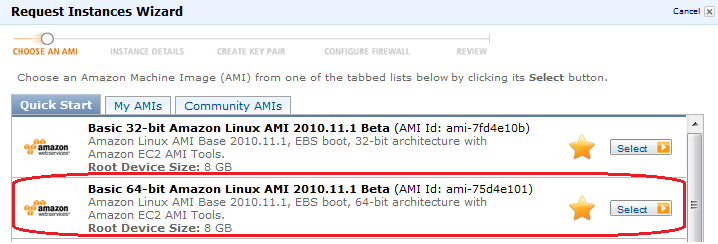
Note that there are no roles or features set up so you do need to set everything up as you want.



Instantiating and connecting to a Linux instance

Embracing the cloud means selecting the best solution for the task which will often mean selecting an open source solution that is optimised to run under Linux. . We advise Selecting the AWS Linux AMI as the base for a number of reasons. We trust Amazon as a source, using an Amazon AMI means updates are applied on start-up, we are comfortable with fedora type commands. Although we will be using Chef to configure our target production Linux instances the important thing about the development environment is to get things up and running quickly. You will have to walk through what you need configuring anyway to allow you to set up your cookbooks and recipes and you are unlikely to have a handle on exactly what the configuration will be till later on in the development cycle so setting up the instances manually is an exercise you will need to go through anyway.

Firstly select the appropriate Linux AMI. We selected the Amazon basic 64 bit Linux AMI. As we are working in the eu region the AMI id we wanted was : ami-75de101



The Linux instances use SSH to make secure connections so the use of SSH keys are important:

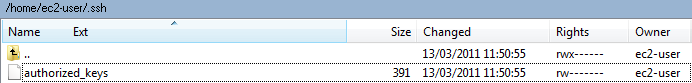


Once you have your key downloaded from the AWS console one of the first things you should do if you are going to be working with Linux instances and using tools like putty and WinSCP is to convert the PEM to a putty private key (PPK) using the puttygen converter .

|  |  |
| --- | --- |
| A key pair is needed to access the instance. This key pair is used to provide authentication when logging onto the instance via SSH. | To create the **KeyPair** use the AWS console |
| After creating the key pair which is a **PEM** a **PPK** Putty compatible key will be needed to allow authentication via SSH | |
| Convert the **PEM** created via the AWS console to a **PPK**. use **PuttyGen** to load the **PEM** |  |
| Click “save private key” click yes at the pop up message box |  |
| Type in the name of the PPK generated key This should ideally reflect the name of the **PEM** it is being generated from |  |

Once the instance has started you can then connect via putty and WinSCP to modify your instance appropriately. Using the elasticfox plugin means you do not need to cut and paste the public DNS of the instance and just need to locate your ppk file when prompted to do so.

The Amazon AWS Linux instance starts from a least privileged account user called ec2-user. This means you will need to elevate when logged onto the instance when you need to modify system files. Access to the instance via SSH requires a public/private key pair. To allow people with different keys from the one used to instantiate the instance the public key needs adding to the authorized keys file for the ec2-user



Use puttygen to open the private key (.pkk) obtain the public key and paste this into the authorized keys file making sure you indicate the key pair name at the end as per the existing line that should be there.



Windows target AMI

This is an AMI that has minimal configuration. The decision as to what is on this AMI should be based on how quickly the windows instance can be spun up and deployed into the target environment. Unlike a Linux instance which can be up and running with deployed applications within 5 minutes from scratch, on first instantiating a windows instance from an Amazon base AMI it can take 20 minutes. Then there is the one time password issue to get past. It will still take considerably longer than instantiating a Linux instance even when you have prepared base windows AMI but it will cut the deployment time none the less by doing a bit of prep work up front.

We decided to use Chef as our configuration tool of choice and after trial and error decided the best approach to using chef with windows on AWS Was to have ruby with the chef gem installed on the target windows AMI. In addition we installed a small .NET application used to download build artefacts from AWS S3. Once you are satisfied with the base windows instance a base AMI is created from the instance.

The Developers AMI

Starting from Amazons base windows 2008 AMI create a new instance that you will need to install the development tools of choice. In our case our development AMI consisted of Visual studio 2010 with the ankh svn and Jira Studio for visual studio plugins . In addition we installed tortoise svn, notepad++ and some other handy utilities. Talk to your developers find out what they want and need and make sure they are available on the base development AMI. Once you have installed all the core development tools. Shutdown the instance then create a new AMI based on this instance.

Linux development target AMI

You will need to install the target applications you will need. In our example we installed Tomcat via the AWS repository using yum we then deployed solr to the tomcat container and configured the schema and database configuration files. We then stopped the instance and created a base AMI using this instance.

RDS configuration

If the solution needs RDS then this will need configuring. There is a walkthrough on creating RDS database instances in the section on Setting up RDS.

S3 Configuration

Grant access to the developer IAM users to use S3 and set up buckets for their use. We cover S3 configuration and security in some detail elsewhere in this book.

Configuration of Other AWS assets

The developers should be granted the ability to create Queues and simpleDB domains. Other assets such as ELB’s should be set up by the administrator of your AWS environment. It’s unlikely these other assets will be needed in the earlier sprints but as the application evolves so will the development environment .

Part of moving to the cloud is encouraging the team to embrace the cloud. In our experience the flexibility and ‘power’ it gives can be over whelming and there is often a sense of clinging to the traditional practises so it is imperative to get the team to start accepting new ways of working. One of the ways to help them along his journey is to get the development environment up and running as quickly as possible. This will demonstrate that the platform is no longer a bottle neck as well as providing the proof that the cloud can deliver faster than anything a traditional environment can.

The first environment to be made available is the development environment. Typically this will consist of a set of AMI’s (Amazon machine image) that can be spun up on demand and the provisioning of the ALM toolset (source control, build services and task management).

Working with instances derived from a base AMI is just like working with a base operating system installed on your workstation or a virtual guest on say VMware or Hyper-v.

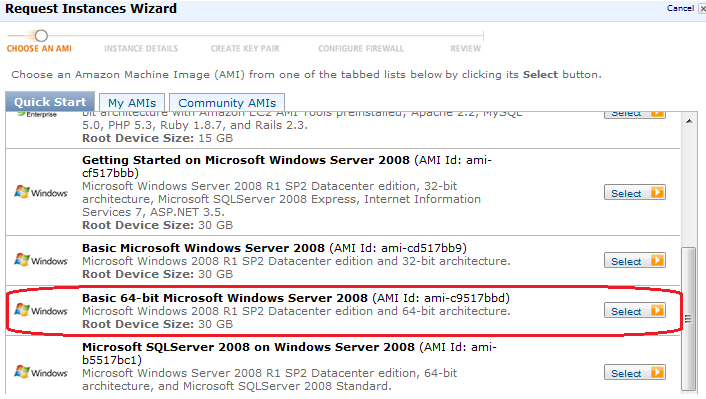
You will need to set up your security group to allow RDP and SSH connections from specified source IPs.

Instantiating and connecting to a Windows instance

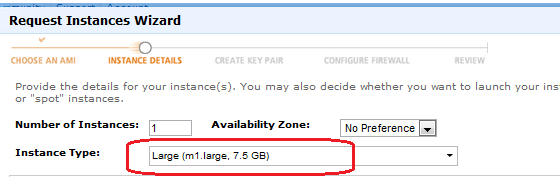
As this is a book about developing .NET applications on AWS it is inevitable that you will be creating some base windows AMIs.

Firstly select the base AMI. We shall select the Amazon windows 2008 x86\_64 AMI (Rumoured at the time of writing that windows 2008 R2 would soon be available)

In our case we are working in the EU region so the AMI we wanted had the AMI id: ami-c9517bbd



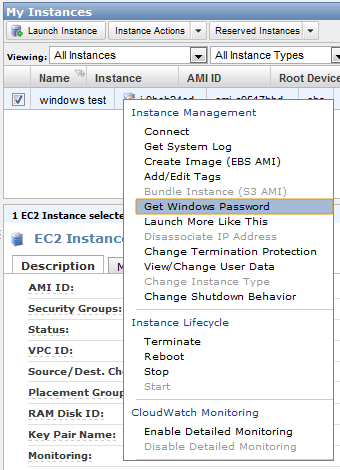
Running a windows instance any smaller than large can prove to be challenging as it is very slow. Using t1.micro is fine once it’s already been configured if you have low throughput web applications or services.



Although you will be using RDP to connect to your windows instance you will need to use your SSH key you created earlier as part of your AWS set up activities.



On first starting a windows instance from an Amazon base AMI you will need to retrieve the administrators password and will need to use your SSH key to achieve this.



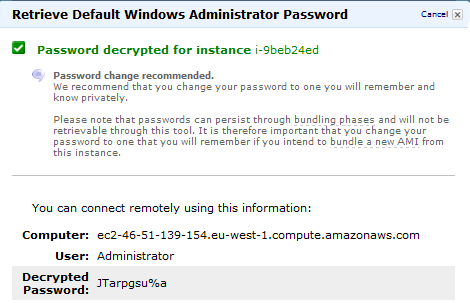
This will be needed for the decryption of the initial Administrators password. Note the warning if you try and retrieve your password before it is ready. This is an ideal time for that coffee break.



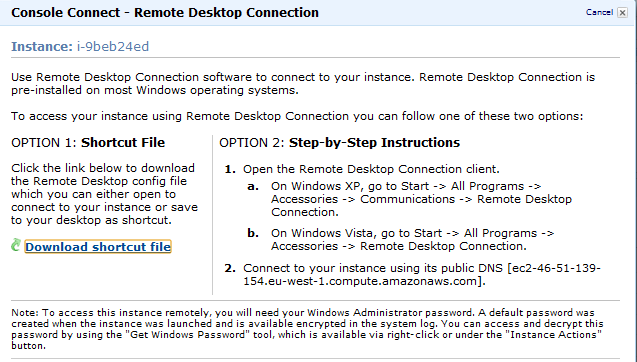
When the instance is ready you will then be prompted to decrypt the password



Use a text editor to open your private key and copy the key into the box click the Decrypt password button and then you will be provided with the randomly generated password.

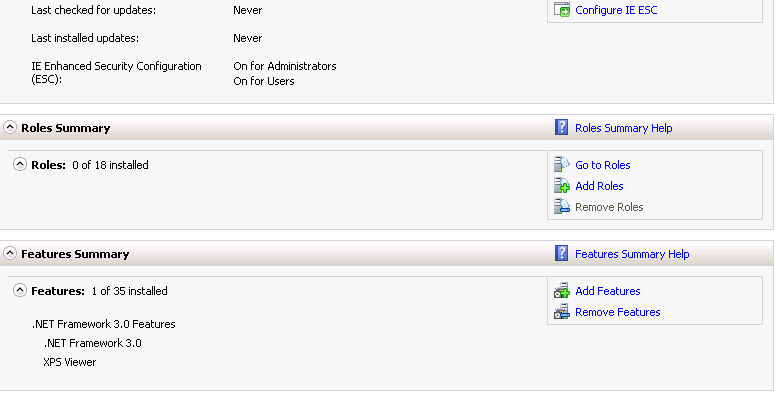


After decrypting you will then want to connect via RDP to your instance and change the administrators password to something you can remember ( Do please use good password selection criteria making sure you use a mix of upper & lower case characters, numbers and non alpha numeric characters and of course keep it reasonably long) . If you select connect from the AWS console you will see the following (They do need to update this information to reflect the increasing use of Windows 7)



Personally I have found connecting via the elasticfox Firefox plug in makes life simpler as you do not have to do the cutting and pasting of the public DNS value but can connect directly to the instance via RDP from the elasticfox plugin

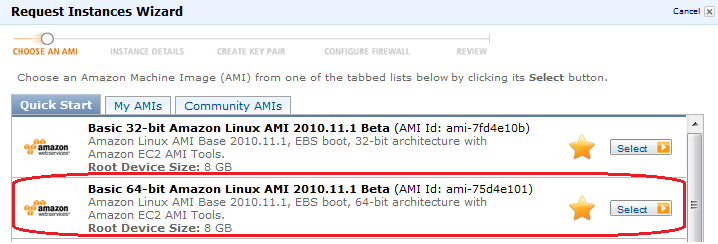
Note that there are no roles or features set up so you do need to set everything up as you want.



Instantiating and connecting to a Linux instance

Embracing the cloud means selecting the best solution for the task which will often mean selecting an open source solution that is optimised to run under Linux. . We advise Selecting the AWS Linux AMI as the base for a number of reasons. We trust Amazon as a source, using an Amazon AMI means updates are applied on start-up, we are comfortable with fedora type commands. Although we will be using Chef to configure our target production Linux instances the important thing about the development environment is to get things up and running quickly. You will have to walk through what you need configuring anyway to allow you to set up your cookbooks and recipes and you are unlikely to have a handle on exactly what the configuration will be till later on in the development cycle so setting up the instances manually is an exercise you will need to go through anyway.

Firstly select the appropriate Linux AMI. We selected the Amazon basic 64 bit Linux AMI. As we are working in the eu region the AMI id we wanted was : ami-75de101



The Linux instances use SSH to make secure connections so the use of SSH keys are important:

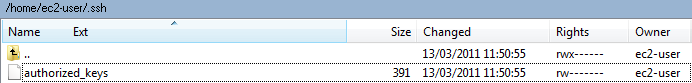


Once you have your key downloaded from the AWS console one of the first things you should do if you are going to be working with Linux instances and using tools like putty and WinSCP is to convert the PEM to a putty private key (PPK) using the puttygen converter .

|  |  |
| --- | --- |
| A key pair is needed to access the instance. This key pair is used to provide authentication when logging onto the instance via SSH. | To create the **KeyPair** use the AWS console |
| After creating the key pair which is a **PEM** a **PPK** Putty compatible key will be needed to allow authentication via SSH | |
| Convert the **PEM** created via the AWS console to a **PPK**. use **PuttyGen** to load the **PEM** |  |
| Click “save private key” click yes at the pop up message box |  |
| Type in the name of the PPK generated key This should ideally reflect the name of the **PEM** it is being generated from |  |

Once the instance has started you can then connect via putty and WinSCP to modify your instance appropriately. Using the elasticfox plugin means you do not need to cut and paste the public DNS of the instance and just need to locate your ppk file when prompted to do so.

The Amazon AWS Linux instance starts from a least privileged account user called ec2-user. This means you will need to elevate when logged onto the instance when you need to modify system files. Access to the instance via SSH requires a public/private key pair. To allow people with different keys from the one used to instantiate the instance the public key needs adding to the authorized keys file for the ec2-user



Use puttygen to open the private key (.pkk) obtain the public key and paste this into the authorized keys file making sure you indicate the key pair name at the end as per the existing line that should be there.



Windows target AMI

This is an AMI that has minimal configuration. The decision as to what is on this AMI should be based on how quickly the windows instance can be spun up and deployed into the target environment. Unlike a Linux instance which can be up and running with deployed applications within 5 minutes from scratch, on first instantiating a windows instance from an Amazon base AMI it can take 20 minutes. Then there is the one time password issue to get past. It will still take considerably longer than instantiating a Linux instance even when you have prepared base windows AMI but it will cut the deployment time none the less by doing a bit of prep work up front.

We decided to use Chef as our configuration tool of choice and after trial and error decided the best approach to using chef with windows on AWS Was to have ruby with the chef gem installed on the target windows AMI. In addition we installed a small .NET application used to download build artefacts from AWS S3. Once you are satisfied with the base windows instance a base AMI is created from the instance.

The Developers AMI

Starting from Amazons base windows 2008 AMI create a new instance that you will need to install the development tools of choice. In our case our development AMI consisted of Visual studio 2010 with the ankh svn and Jira Studio for visual studio plugins . In addition we installed tortoise svn, notepad++ and some other handy utilities. Talk to your developers find out what they want and need and make sure they are available on the base development AMI. Once you have installed all the core development tools. Shutdown the instance then create a new AMI based on this instance.

Linux development target AMI

You will need to install the target applications you will need. In our example we installed Tomcat via the AWS repository using yum we then deployed solr to the tomcat container and configured the schema and database configuration files. We then stopped the instance and created a base AMI using this instance.

RDS configuration

If the solution needs RDS then this will need configuring. There is a walkthrough on creating RDS database instances in the section on Setting up RDS.

S3 Configuration

Grant access to the developer IAM users to use S3 and set up buckets for their use. We cover S3 configuration and security in some detail elsewhere in this book.

Configuration of Other AWS assets

The developers should be granted the ability to create Queues and simpleDB domains. Other assets such as ELB’s should be set up by the administrator of your AWS environment. It’s unlikely these other assets will be needed in the earlier sprints but as the application evolves so will the development environment .

Setting up source control and build services.

As mentioned in the first half of this book we selected the hosted solution from Atlassian Jira Studio. We accept that TFS offers an excellent all round experience but there are plenty of resources covering using TFS as your ALM solution for .NET projects so we will not cover that. we will instead focus on the solution we selected which we believe embraces some of the fundamentals that make working with the cloud so enticing.

Getting started is a simple case of signing up for Jira studio which like AWS is pay as you go. Using the build services does assume you already have an AWS account. The build controller provided is elastic bamboo which in turn uses elastic agents which are AWS ec2 instances that are spun up by the build controller to run the build. So in addition to the costs for Jira Studio you also need to bear the costs for the elastic agents.

Although Jira Studio is not an open source solution it does suffer the same problem all open source tools seem to have in that unless they specifically target windows then windows is definitely a second class citizen. In our case this meant we needed to create a custom windows AMI to be used to create the elastic-agents. Unfortunately there were no available instructions on creating a custom elastic-agent for windows users so we had to use the Linux instructions as guide. Although we got it working it is unsupported by Atlassian but hopefully they’ll be supporting windows soon as from our experience it did actually work.

Once signed up you get access to the following hosted services:

Subversion – source control

Jira – Issue Tracking

GreenHopper – agile project management tool

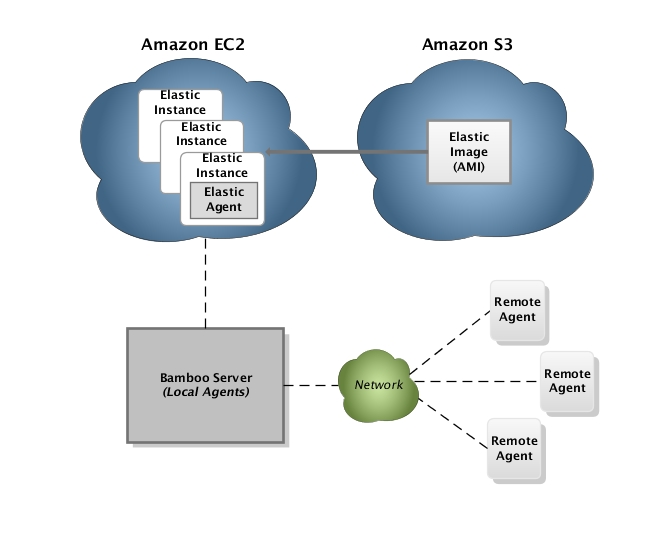
Confluence – Wiki

Bamboo – Continuous integration and release management.

All of which apart from Bamboo are ready to go out of the box once your account has been set up.

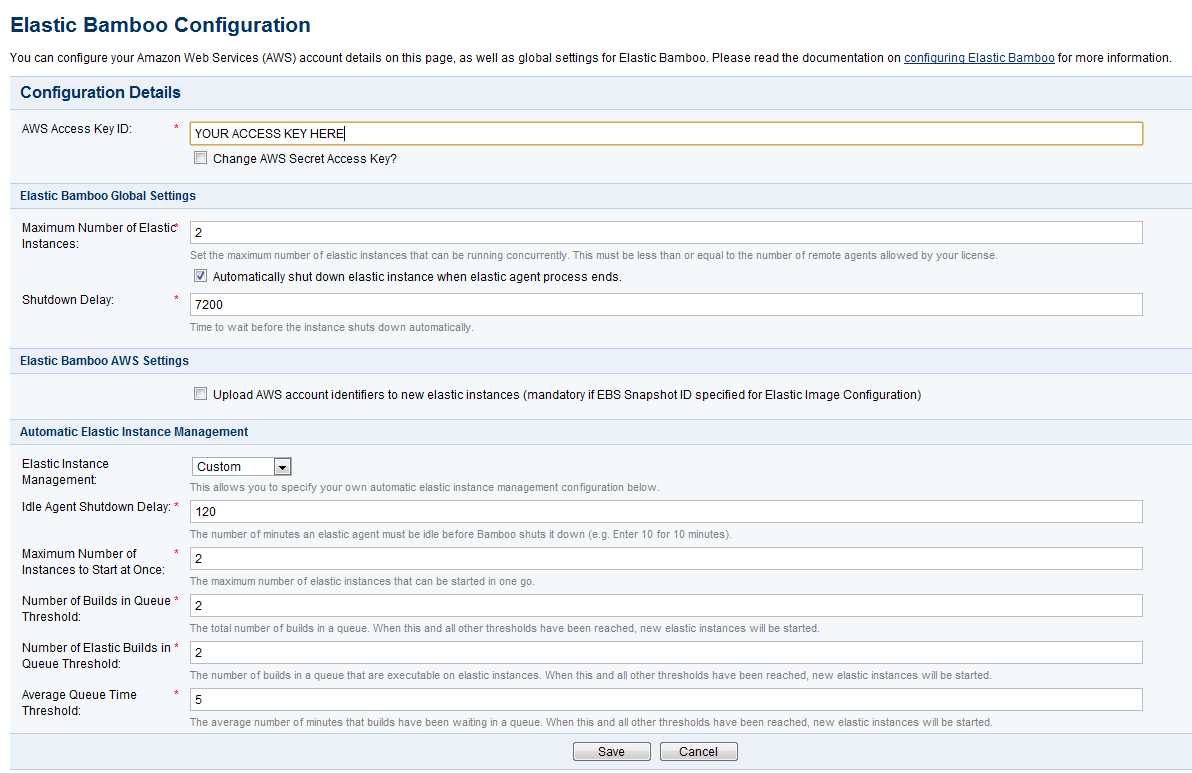
Configuring Elastic Bamboo for building .NET projects

Elastic Bamboo makes use of EC2 instances to carry out the builds. The diagram below from Atlassian illustrates the configuration:

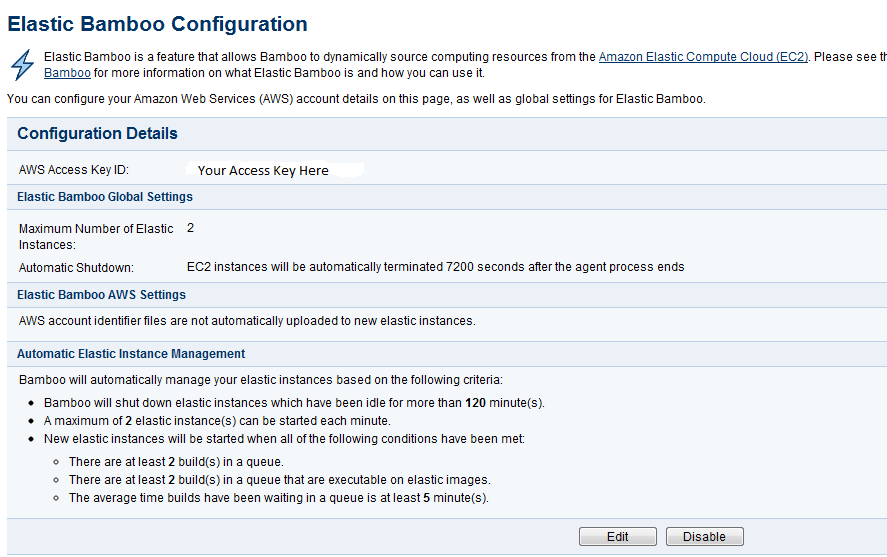


003-27

The first thing to be done is the initial configuration that ties in your Jira Studio Elastic Bamboo configuration with your AWS account. The configuration page also allows you to set up parameters related to the management of how many instances can run at any one time, what triggers an instance start and shut down.



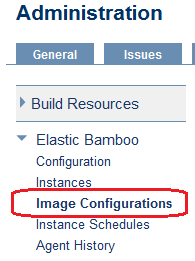
003-30



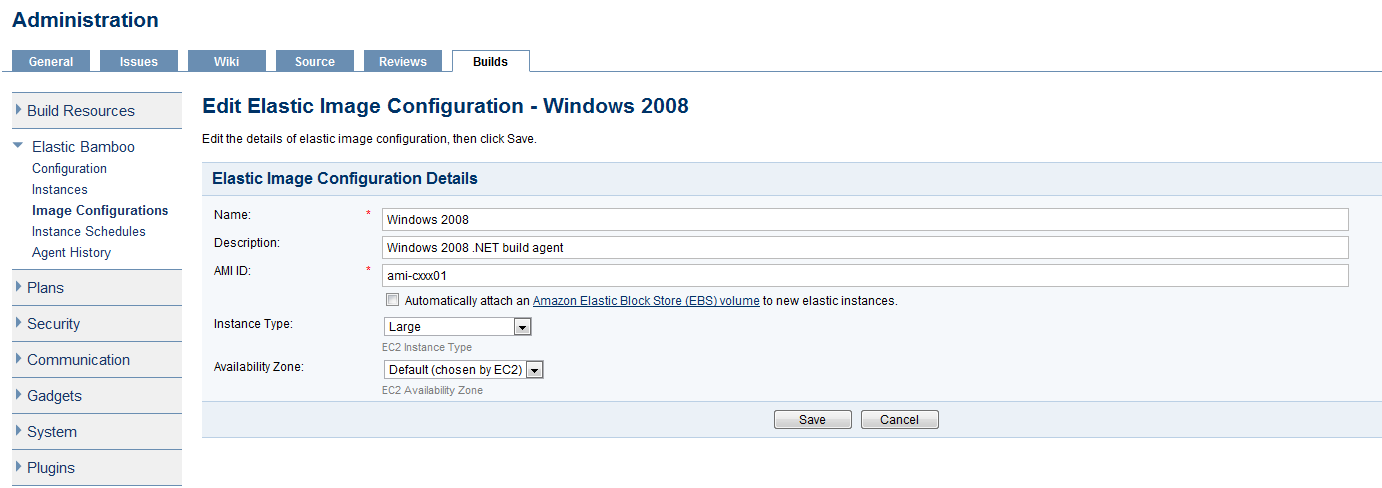
003-28

Atlassian provides a base Linux AMI but windows is not fully supported out of the box for use with Elastic Bamboo the first thing you will need to do is create your base build agent. Instructions for doing this are detailed here: <http://devopscloud.net/2011/03/25/setting-up-a-windows-ami-for-use-with-elastic-bamboo/>

Once you have created the AMI it will need to be registered with Elastic Bamboo by selecting the Image configuration menu item

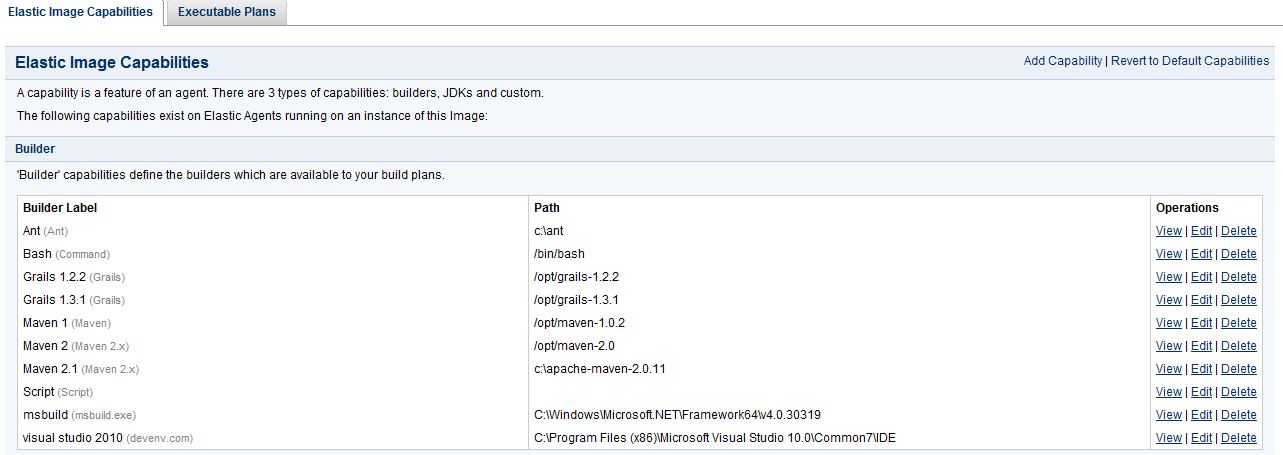


003-29



003-31

The next thing to do is to associate builders with the Elastic Image /AMI. Builders are basically what builds/ compiles your application e.g msbuild , ant etc. You need to locate the location of your builders and add them to the build capabilities of your Elastic Agent/AMI. For .NET projects you typically should add the path to msbuild and visual studio. Elastic Bamboo recognises these as standard builders.



003-32

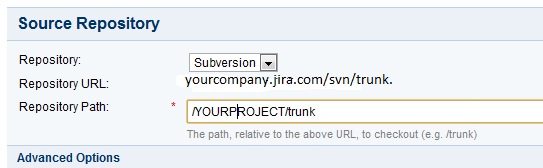
Once you have the capabilities set up the next thing to do is to associate the build plan with the builder

Setting up a build plan involves going through all the tabs and filling in the appropriate details.



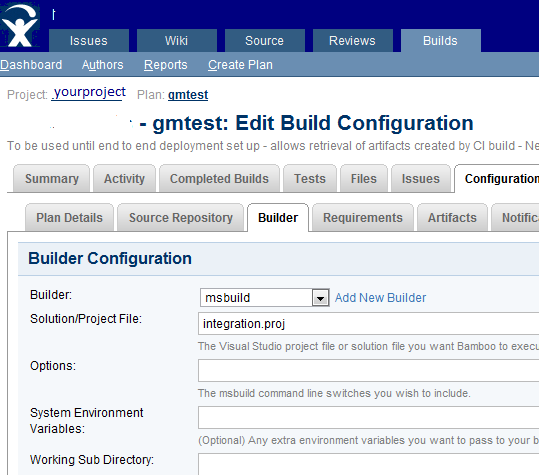
003-35

Jira Studio uses svn so this will be your repository you then need to point it to the repository url for the project



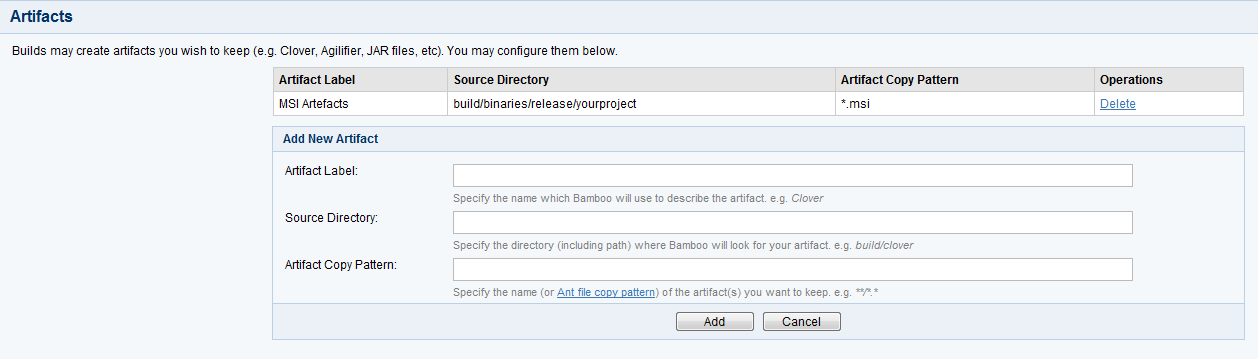
003-36

You then need to select the appropriate builder



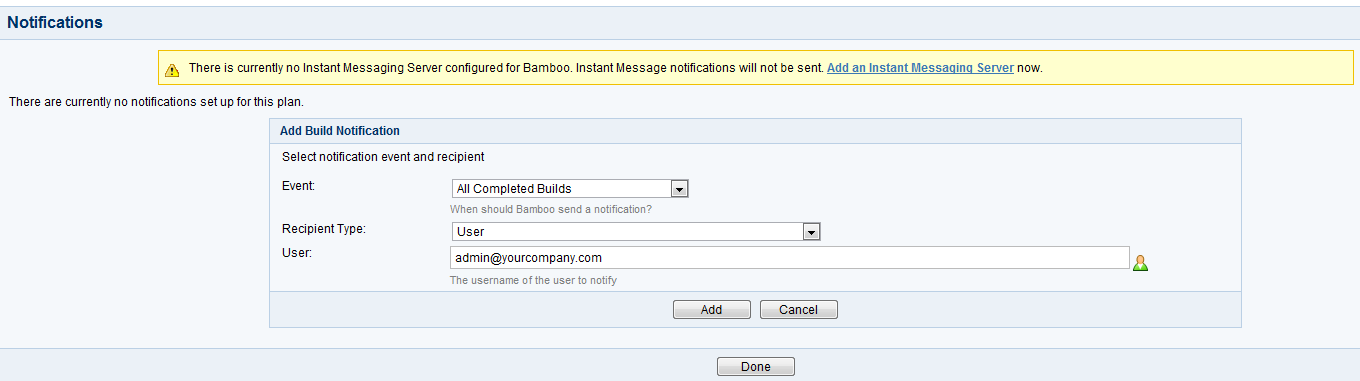
003-33

The artifacts tab is where you add a list of artefacts that are produced by the build . On a successful build you will then be able to download the artefacts directly from Bamboo



003-37

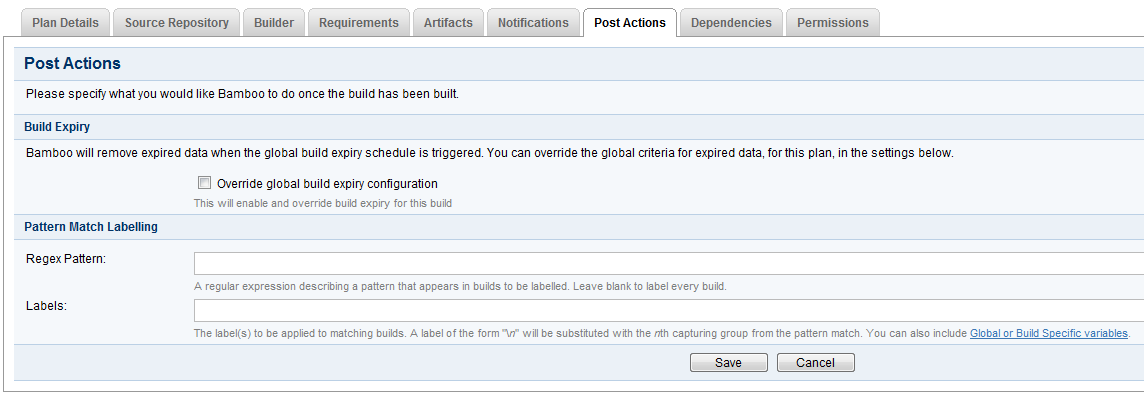
The notifications tab allows you to set up recipient email addresses and integration with Instant Messenger.



003-38

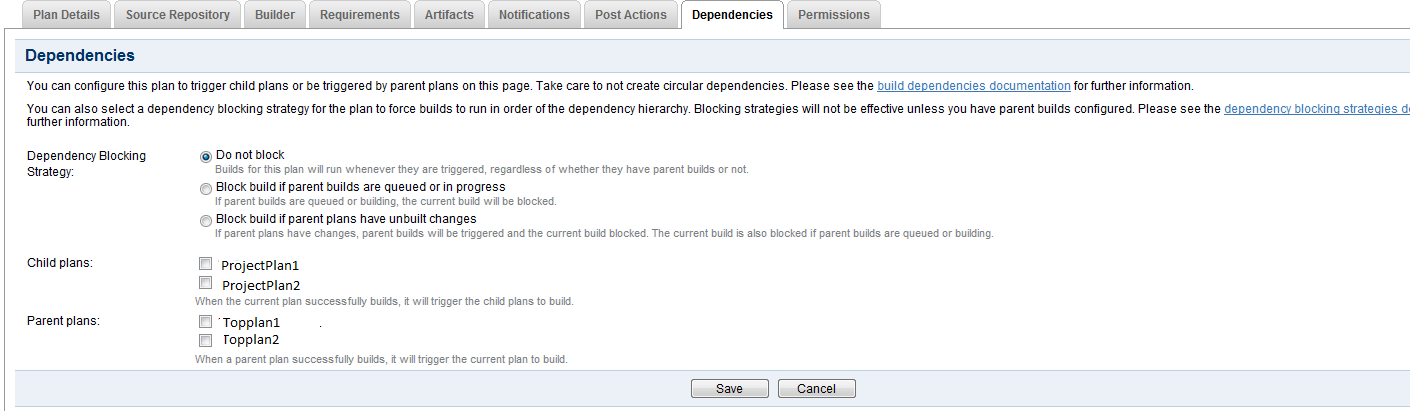
The email address that is being added must exist as a valid user within your Jira Studio configuration.

Post Actions is where you set up what you want to happen after a build.



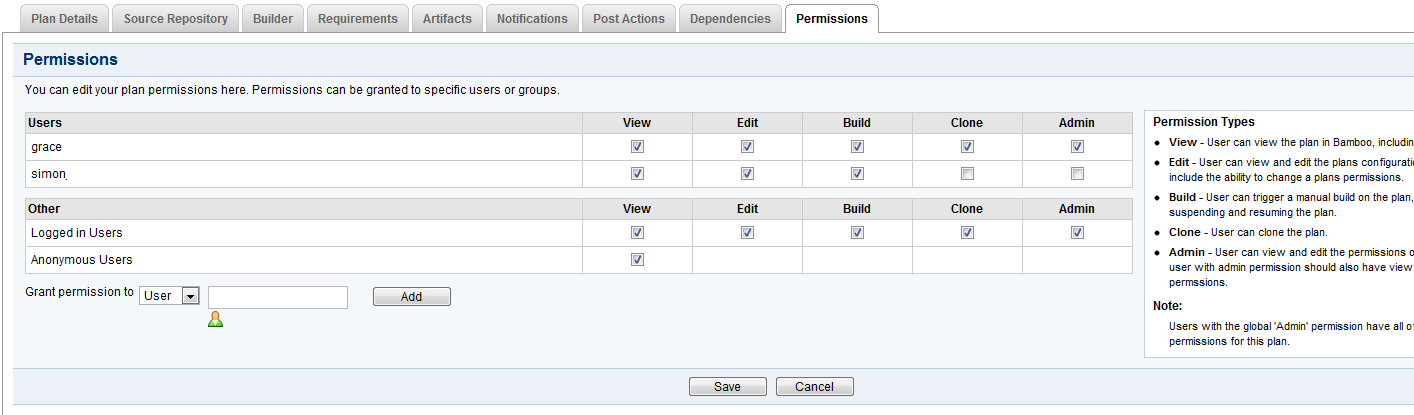
003-39

The Dependencies tab is where you can set up chained builds



003-40

The Permissions tab is where you grant people permissions re what they can do i.e can they just view build results or can they run the builds



003-41

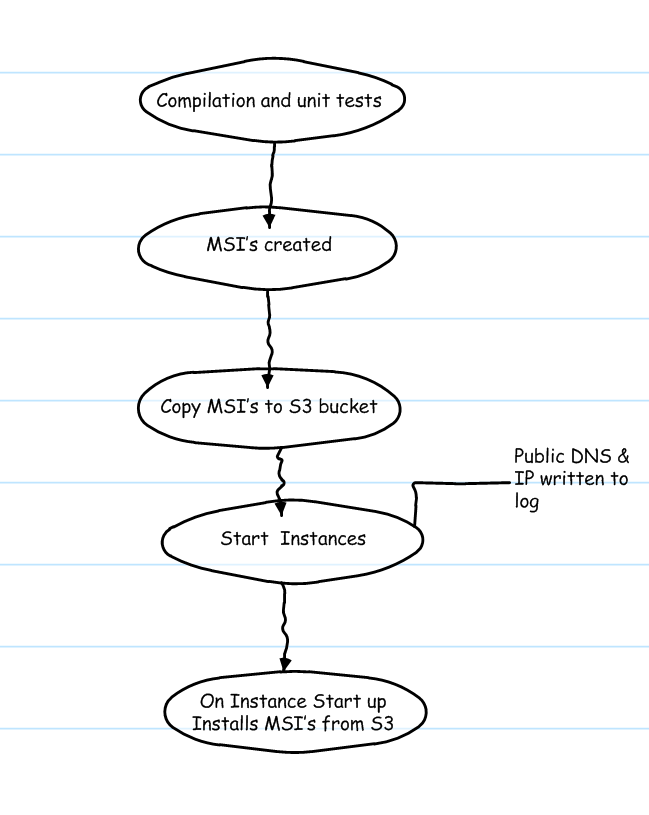
Using AWS resources as part of your actual continuous integration build

AWS provides SDK’s for .NET . Java, ruby, python and PHP . This allows you to write small useful applications that enable you to exploit AWS resources as part of your CI build process.

The AWS SDK for .NET supports the following AWS resources EC2, S3, SimpleDB, RDS, SQS,SNS, CloudFront, Elastic MapReduce, Cloudwatch, ELB and autoscaling.

Uploading build artefacts to an S3 bucket, starting instances and automatically deploying those artefacts are all tasks that can be incorporated as part of the CI build by using the AWS SDK.

The sketch below illustrates the flow of a typical CI build that uses AWS resources as part of the build process



003-77

One of the tasks that will inevitably need to be done is to incorporate the copying of build artefacts such as MSI’s to an S3 bucket.

To achieve this task you will first need to write an application that uses the AWS SDK that allows the upload and download of artefacts from S3. Driving it using an application config file allows you to change bucket targets and the files that need to be transferred without having to recompile the application.

Using the SDK you can also write an application again driven by an application configuration file that allows you to start different instance types and configurations.

There are some examples of using the SDK on the AWS site but you will probably find yourself spending a fair amount of time getting your head round the methods and exact semantics to allow you to use the SDK <http://docs.amazonwebservices.com/sdkfornet/latest/apidocs/Index.html> as the getting started guide barely gets you started ( I’d advise sitting next to a full time .NET developer if coding isn’t your fulltime job ) .

You can make use of windows services or the RunOnce registry key to enable the instance to call a PowerShell script that facilitates the installation of the MSI’s created as part of the build process after the windows instance has started up .

You can also use the build process to spin up an environment using a combination of cloudformation and chef calls.

Using IAM to grant access to resources for your AWS account.

IAM (Identity and Access Management) allows you to create users and groups to grant access to resources accessible via your AWS account.

To start using IAM the first thing to be done is to define the types of users needed and what access they will need to be given; then create a matrix of what access they will need to what AWS resource. This will help create the IAM policy files.

Using groups is the appropriate way to grant access to resources for IAM users. Once you have created the matrix, you will be in a position to identify what groups will need to be created and what policies will need to be applied to those groups.

IAM uses JSON policy files to apply permissions for resources. These policies are attached to users and groups.

As well as providing access to AWS resources using the CLI and API , access can also be granted to the AWS console to IAM users.

To illustrate how to actually configure IAM the matrix below shows 3 groups we will walkthrough how to use IAM polices to grant access rights to each group.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **admin group** | **partner group** | **dev group** |
| **S3** |  |  |  |
| **Create Bucket** | √ |  |  |
| **Delete Bucket** | √ |  |  |
| **Add resources** |  | √ |  |
| **Delete resources** |  |  |  |
| **Manage IAM** | √ |  |  |
| **AWS Console** |  |  | √ |
| **view instances** |  |  | √ |
| **start instances** |  |  | √ |
| **stop instances** |  |  | √ |

The following users are members of the groups indicated:

|  |
| --- |
| Simon member of the admin group  Colin member of the dev group  Jacqui member of the partner group |

Now that we know what resources we need to grant to who the next thing to do is to make sure your AWS shell is set up to use the IAM command line tools ( Hopefully you will have done this when setting up your AWS shell )

Start the shell and create the three groups by using the IAM-groupcreate command:

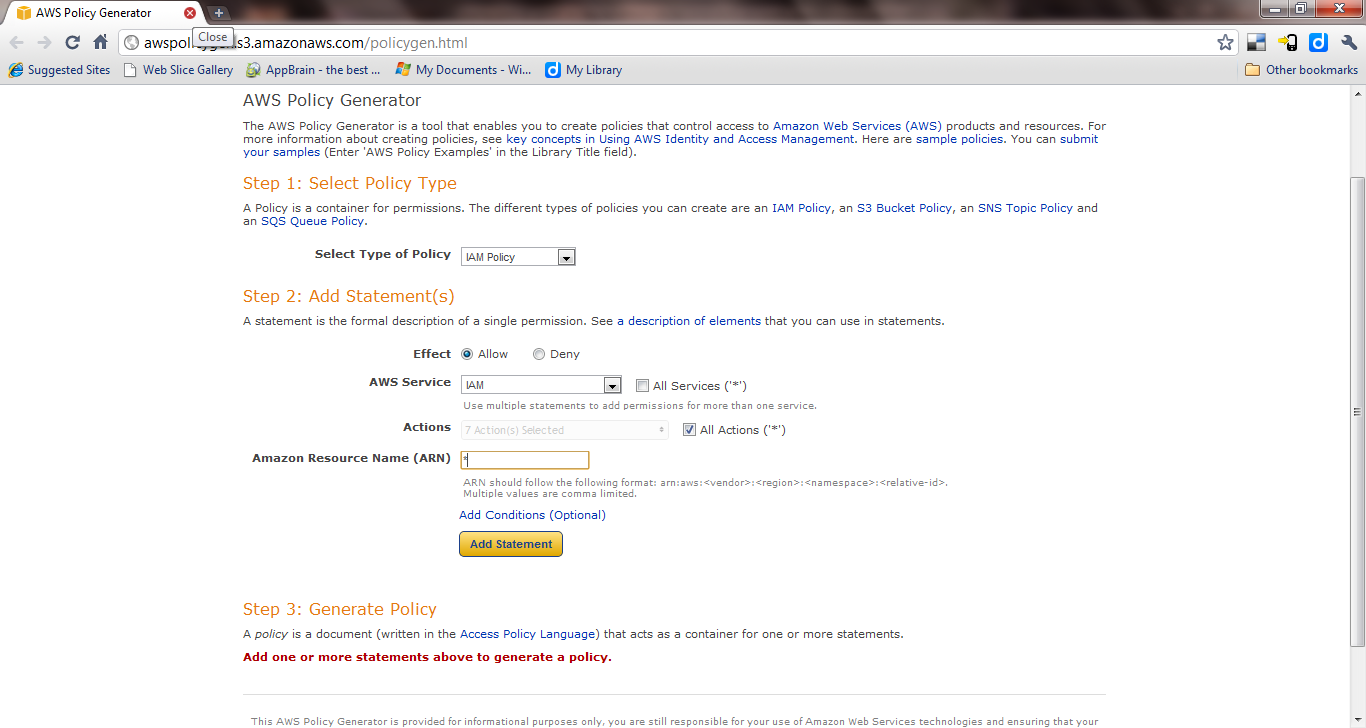
iam-groupceate –g admin

iam-groupcreate –g partner

iam-groupcreate –g dev

Next create the JSON policy files to be used for each group. This can be done using the JSON stmt generator provided by AWS, a JSON editor ( or even notepad) or one of the 3rd party tools.

We will walkthrough using the AWS policy generator first which is located here: <http://awspolicygen.s3.amazonaws.com/policygen.html>

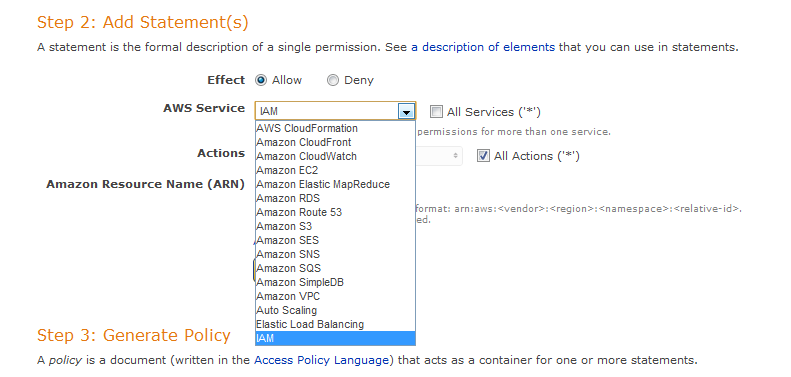


003-42

From here you can generate policies for IAM, S3 bucket, SNS topic and SQS queue policies .

The first policy to be created is the IAM policy that allows managing the other users

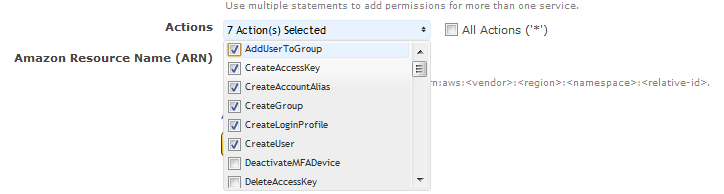
Next select the service from the AWS service drop down



003-43

Select IAM

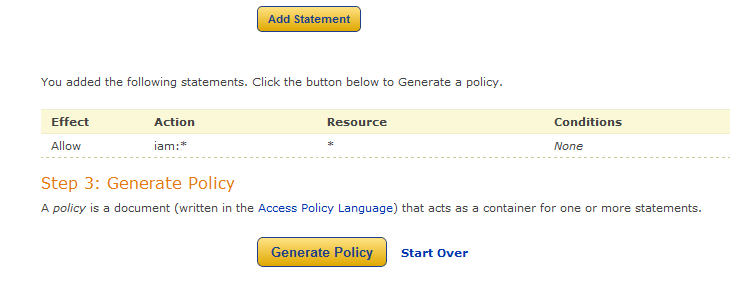
You can select individual actions action you wish to be granted by checking alongside the appropriate action in the action drop down list



003-44

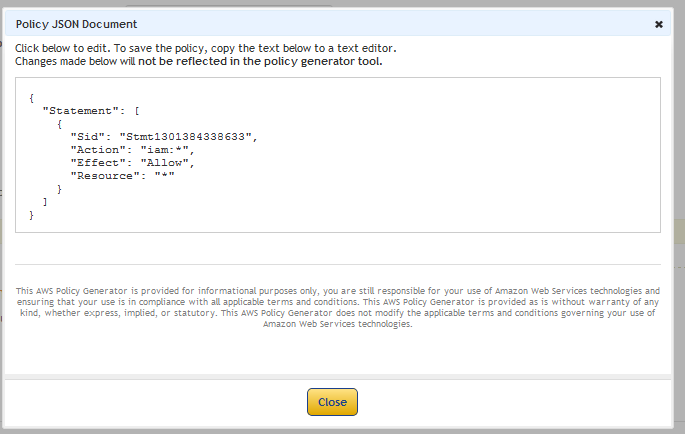
In our case though we want Simon to have full control over the management of iAM users so we will select All Actions.

Then when all the required actions have been completed click the Add statement button



003-45

This then gives you the option to generate the policy by clicking the generate policy button



003-46

As you can see the JSON is easy to understand.

The statement is then copied to a text file and saved for use later.

For ease of use store all the policies in a specific Policy folder. These policy files should be added to your source control system (in my case this is githhub)

Policies are additive so we like to create separate policy files for each AWS resource and apply them separately.

The admin s3 bucket policies and the dev group ec2 policies should be generated the same way. We’ll come back to the console policy later.

Once the policies have been saved as text files the next action is to upload the policies attaching then to the appropriate group. This is achieved by using the iam-groupuploadpolicy command.

So the two upload commands for the admin group are:

iam-groupuploadpolicy -g admin -p AdminIAMPolicy -f adminIAMpolicy.txt

iam-groupuploadpolicy -g admin -p AdminS3Policy -f admins3bucketpolicy.txt

In the above commands the value after –p refers to the name you assign the policy being uploaded and the value after the –f refers to the actual file the policy is saved in. it’s important that you can associate the policy name with the policy uploaded so having naming convention makes sense here ( It’s always good to have a naming convention ).

You can verify what policies are associated with an IAM group by using the iam-grouplistpolicies command against each group.



003-47

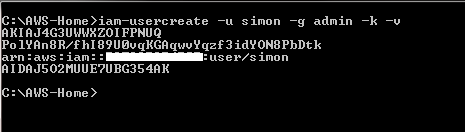
With a couple of groups set up the next thing to do is to create some users and add them to the groups. A quick reminder of who belongs to which group:

|  |
| --- |
| Simon member of the admin group  Colin member of the dev group & AWS console access  Jacqui member of the partner group |

IAM users are created using the IAM-usercreate command:

iam-usercreate -u simon -g admin -k –v

The above creates the user Simon adds him to the admin group, creates an access key for him. The –v stands for verbose.



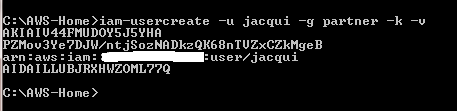
003-48

The first two lines of the file are the user’s access key id and secrete access key. These should be saved in a user specific credential file as the user will need these to use the resources they have been granted access to. For Simon to then be able to access the command line tools he will need to set up an AWS shell as discussed in the section on setting up the AWS shell but in his case you set up the following line in the batch file : set AWS\_CREDENTIAL\_FILE=C:\AWS-Home\credential-file.txt to point to Simons credential file you created earlier. Simon’s credential file has the following two lines:

AWSAccessKeyId=AKIAJ4G3UWWXZOIFPNUQ

AWSSecretKey=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

Creating Jacqui is similar to creating Simon



003-49

Now we have two IAM users we’ll check that they do actually only have the access rights conferred on them by the policies assigned to the groups they are members of.

When Simon uses his shell and types iam-listusers he can list all the iam users associated with the AWS account



003-53

Jacqui however is denied access



003-54

We still have another user to create.



003-50

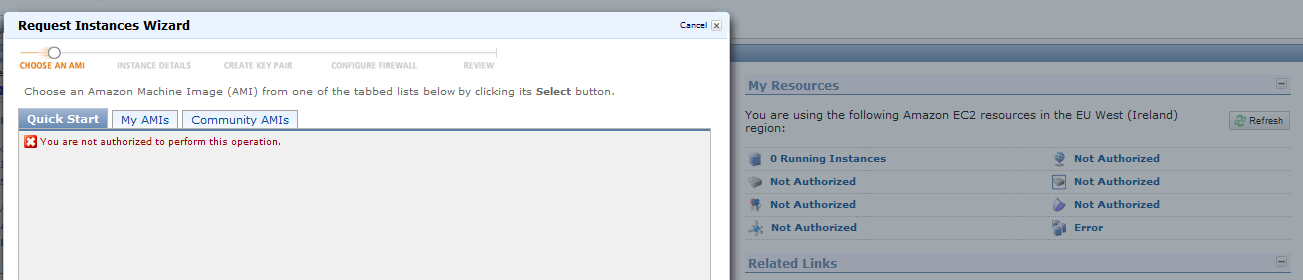
But we’re not quite finished setting up Colin though as the second step in creating Colin is granting him a login profile with a password of ‘HelloAWS’ which will grant him access to the AWS console.

Access for an IAM user to the console is granted via a URL similar to :

<https://xxxxxxxxxx.signin.aws.amazon.com/console/ec2>

Where xxxxx is your AWS account ID.

But when Colin tries to use the ec2 resources after logging onto the console and ties to start an instance he will be denied the ability to do so.

****

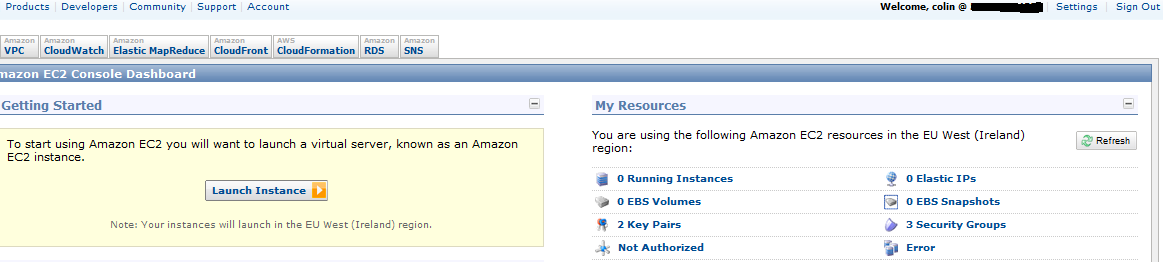
003-52

The login profile will need to be granted access via a policy of it’s own so as with the IAM configuration we have just walked through we need to add a policy to the group Colin belongs to .

The following JSON is used to create the policy for allowing the dev group members access to ec2 facilities (This can obviously be fine tuned down to prevent certain actions as required.

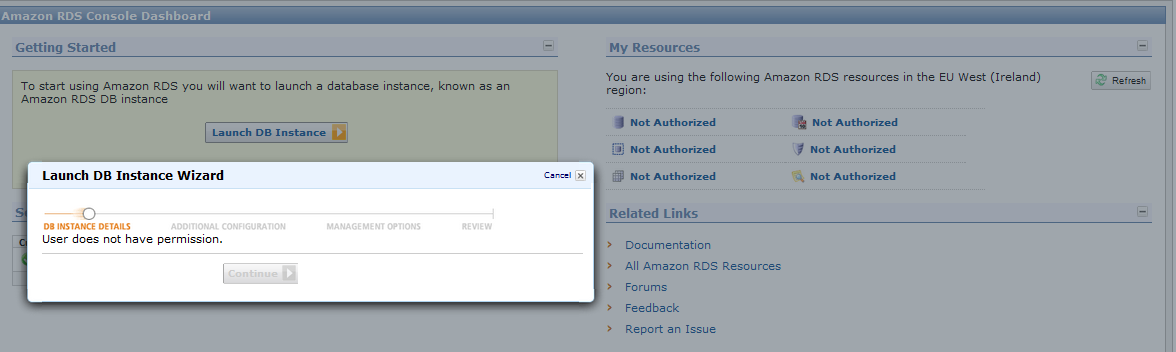
|  |
| --- |
| {  "Statement": [  {  "Sid": "Stmt1299620042038",  "Action": "ec2:\*",  "Effect": "Allow",  "Resource": "\*"  }  ]  } |

Now when Colin logins in after the policy has been applied he can start instances



003-55

But as he has only been given access to ec2 so when he strays to look elsewhere access is denied



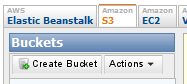
003-57

Now that we’ve gone through setting up IAM users and granted them access to resources via the command line the timely task to tackle next would be setting up access to S3 buckets and objects.

**Configuring S3**

Creating buckets can be achieved easily by using the AWS console or a 3rd party product like S3 browser. The bucket needs to be created in a specific region.

To use the AWS console click the S3 tab then click the create Bucket button



003-61

Name the bucket and select the region.

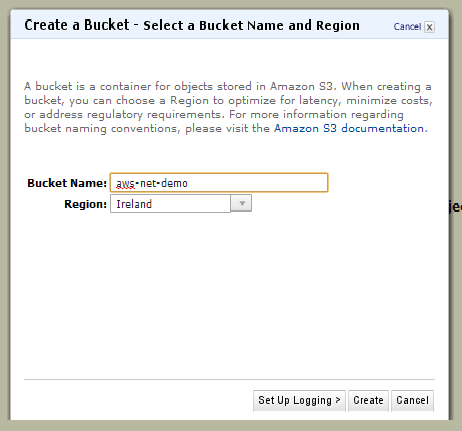
A bucket name must be unique across all existing bucket names in Amazon S3.

Bucket names must comply with the following requirements:

* Can contain lowercase letters, numbers, periods (.), underscores (\_), and dashes (-)
* Must start with a number or letter
* Must be between 3 and 255 characters long
* Must not be formatted as an IP address (e.g., 265.255.5.4)

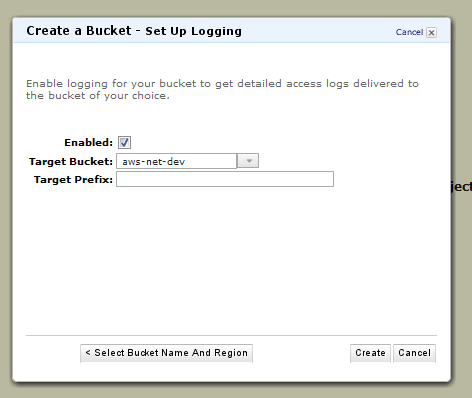
To conform with DNS requirements, AWS recommend following these additional guidelines when creating buckets:

* Bucket names should not contain underscores (\_)
* Bucket names should be between 3 and 63 characters long
* Bucket names should not end with a dash
* Bucket names cannot contain two, adjacent periods
* Bucket names cannot contain dashes next to period
* The location of an object in Amazon S3 is a URL, generally, of the form: http://[bucket-name].S3.amazonaws.com/[key] .



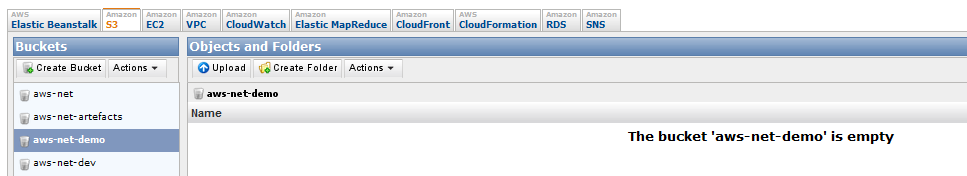
003-62

At this stage it is also possible to set up logging.



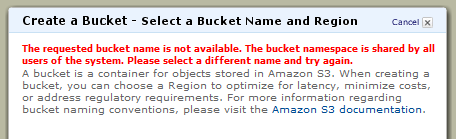
003-63

This provides access to the newly created bucket.



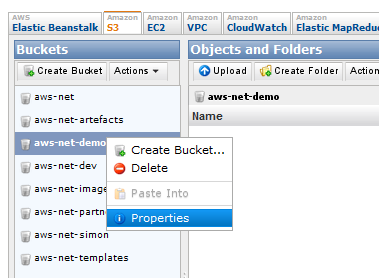
003-65

You must define a bucket naming convention as early in the project as possible and create all your buckets up front. Creating them programmatically can be inadvisable as you cannot guarantee that you will be able to name the buckets with the name you anticipated. Using the console to create a bucket name that already exists within S3 will produce an error:



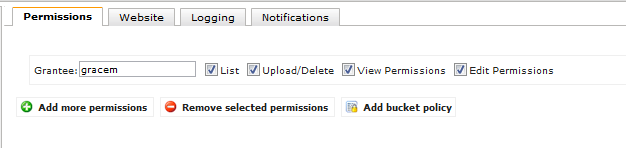
003-64

Right clicking on a bucket from the AWS S3 console gives access to the properties menu item



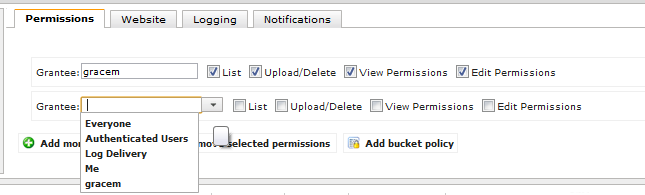
003-65

On first creating a bucket using the console the AWS account owner is the only one who has access to the bucket. The permissions tab is where permissions via ACL’s can be set up



003-66

To add another ACL Click on the Add more permission button and select the Grantee dropdown list.

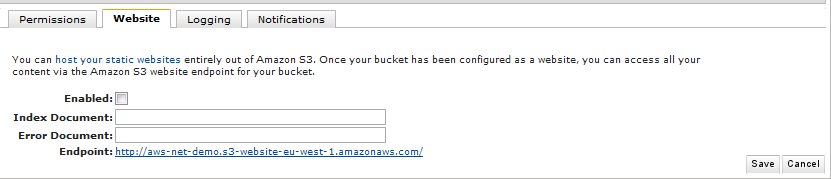


003-68

From here You can grant access to everyone (Anonymous access) and also Authenticated users (Any AWS account using the email they signed up with AWS) . There is also a log delivery grantee.

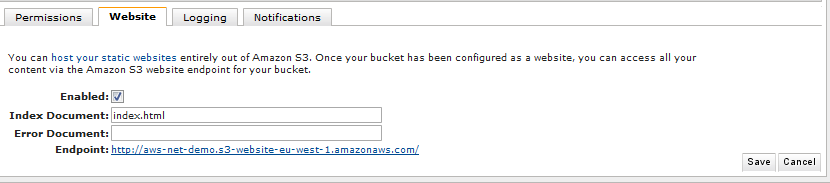
You can set the permissions shown.

* List Grants the ability to list the objects in the bucket.
* Upload/Deletegrants the ability to upload and delete objects ( use with caution)
* View Permissions grants the ability to view the permissions associated with the object
* Edit Permissions grants the ability edit the permissions associated with the object



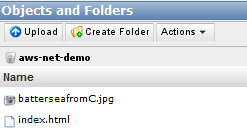
003-69

The website tab is a neat feature as it allows you to set up a static website using S3 with no need for running a web server. It is easy, and cost effective. To do this click on enabled, enter an index document upload the index document and any other pages or images. Set the bucket policy (Bucket policies are covered in detail later in this chapter) to allow public read access.



003-71

We have put two objects in the bucket we use as a static website :

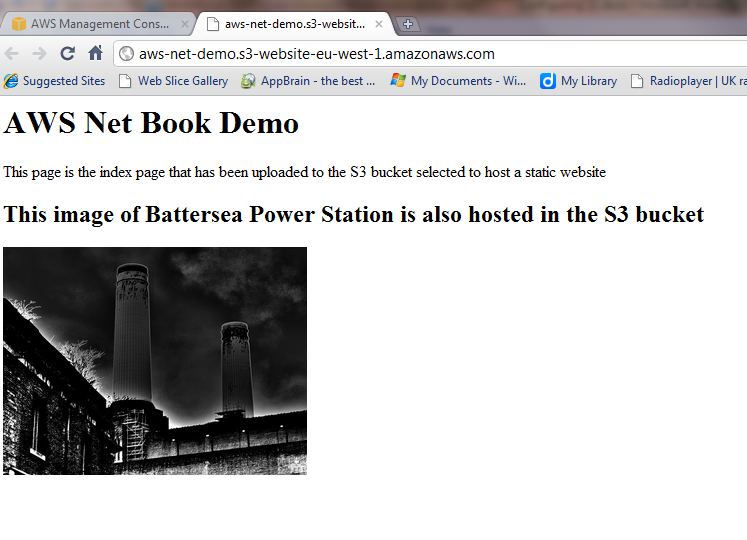


003-72

The bucket policy applied is

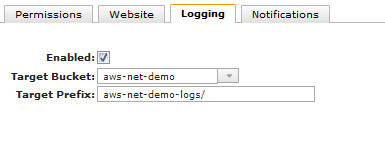
|  |
| --- |
| {  "Statement": [  {  "Sid": "AllowPublicRead",  "Effect": "Allow",  "Principal": {  "AWS": "\*"  },  "Action": [  "s3:GetObject"  ],  "Resource": [  "arn:aws:s3:::aws-net-demo/\*"  ]  }  ]  } |

That is all that is involved with getting a static web site up and running and when you click on the website endpoint you can see the simple page we created.



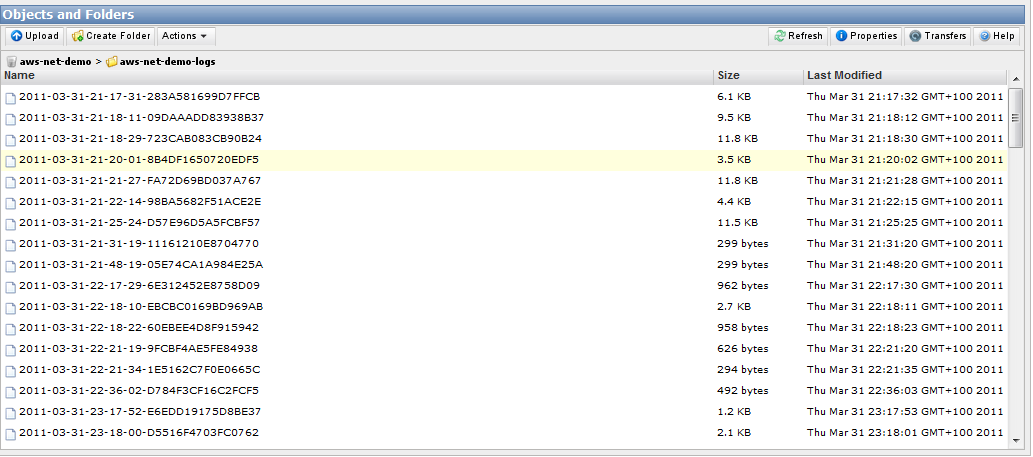
003-70

The logging tab is where you set up the ability to gather detailed access logs. Click the enabled tab and select from the drop down list the bucket you wish to collect access logs for . Enter an appropriate target prefix. In the example below access logs for the bucket aws-net-demo are written to the ‘folder’ aws-net-demo-logs/



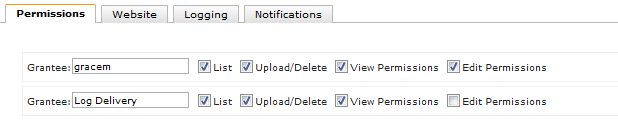
003-73

The access logs are then written to the folder aws-net-demo-logs in the bucket aws-net-demo. These logs do incur storage costs so you do need to monitor this folder and think carefully about whether you really need logging on. For heavily accessed buckets the logs can soon grow



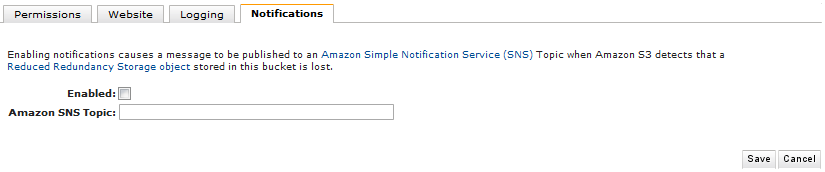
003-76

When you click back to the permissions tab. You will notice that after setting up logging the Log Delivery service has been automatically set up .



003-74

The last tab is where Simple Notification Service (SNS) is set up. It is currently only available to be used to indicate when a reduced redundancy storage object is lost. ( We cover SNS configuration in chapterxx)



003-75

Objects are accessible via an endpoint of the form The location of an object in Amazon S3 is a URL, generally, of the form: http(s)://[bucket-name].S3.amazonaws.com/[key]

It is highly likely that you will want to use your company or product identity as part of the endpoint and that can be achieved by the use of CNAMES mapping to the target bucket. You need to own the domain. It is important that when it comes to naming the bucket this will need to match the CNAME.

As an example we will be using the domain: aws-net-book.net and the target bucket pictures.aws-net-book.net

The came record needs to look like this:

**Pictures.aws-net-book.net CNAME pictures.aws**-**net-book.net.s3.amazonaws.com**

Once set up you can then access your bucket objects using the CNAME.

**S3 resilience**

S3 is designed for mission-critical and primary data storage. Objects are redundantly stored on multiple devices across multiple facilities in an S3 Region. Objects are synchronously written to multiple locations. The level of redundancy built into S3 means that AWS are able to offer SLA’s that provide 99.999999999% durability and 99.99% availability of objects over a given year. If this level of durability and availability is not needed then there is also the option of using reduced redundancy which provides  99.99% durability.

Versioning is also available which provides the ability to recover from accidental deletions

**Setting up S3 bucket policies and ACL’s**

We touched briefly on S3 security when looking at how to set up buckets, but S3 security can be confusing and the best way to understand the use of ACL’s and Bucket policies is to walk through a few examples. To do this we will use what we have found to be one of the most useful tools to help manage S3 ‘S3 browser’ (These guys have done an awesome job and release features at almost the same rate of knots as AWS themselves. Investing in the Pro version will be worth it)

S3 will inevitably be an integral part of your application and getting your head around setting up ACL’s and bucket policies is an important task.

**Using IAM and bucket policies**

We will be using the scenarios we described in the chapter on ‘Using IAM to grant access to resources for your AWS account’ .

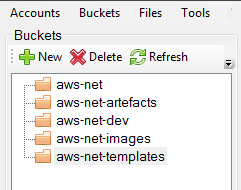
Just a quick reminder as to what needs setting up for S3:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **admin group** | **partner group** | **dev group** |
| **S3** |  |  |  |
| **Create Bucket** | √ |  |  |
| **Delete Bucket** | √ |  |  |
| **Add resources** |  | √ |  |
| **Delete resources** |  |  |  |

And which user is in which group:

|  |
| --- |
| Simon member of the admin group  Colin member of the dev group & AWS console access  Jacqui member of the partner group |

We have the following buckets set up:



003-01

The free version of S3 browser allows you to set up two users so we’ll first add Simon using the access key ID and secret

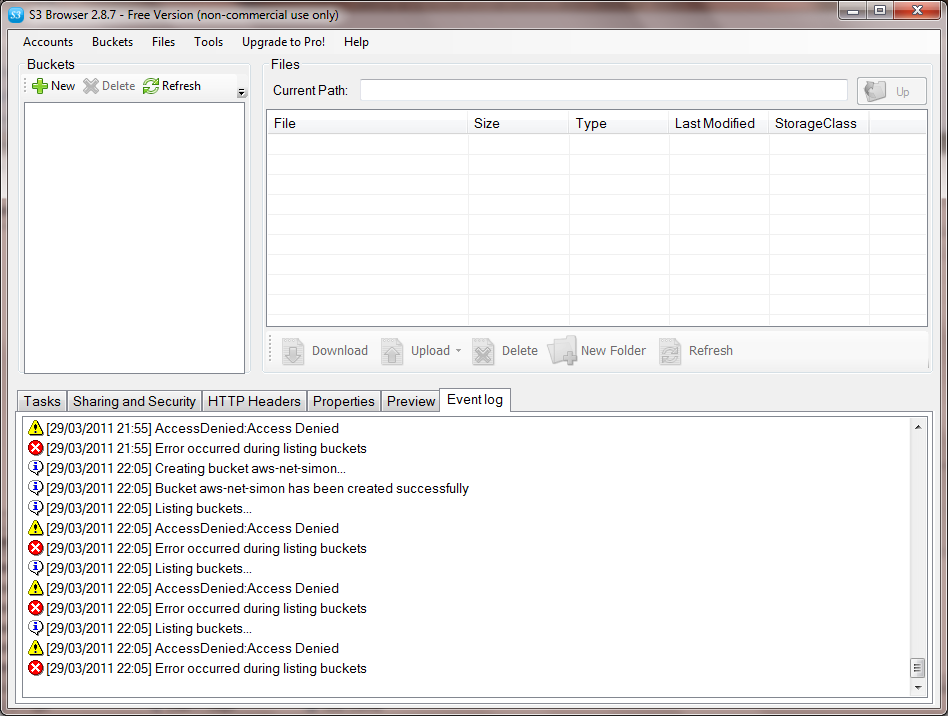
We have set up the IAM users as per the walkthrough on using IAM.

A quick reminder of what the JSON used to create the S3 policy applied to the admin group looks like:

|  |
| --- |
| {  "Statement": [  {  "Sid": "Stmt1301245213618",  "Action": [  "s3:CreateBucket",  "s3:DeleteBucket"  ],  "Effect": "Allow",  "Resource": "\*"  }  ]  } |

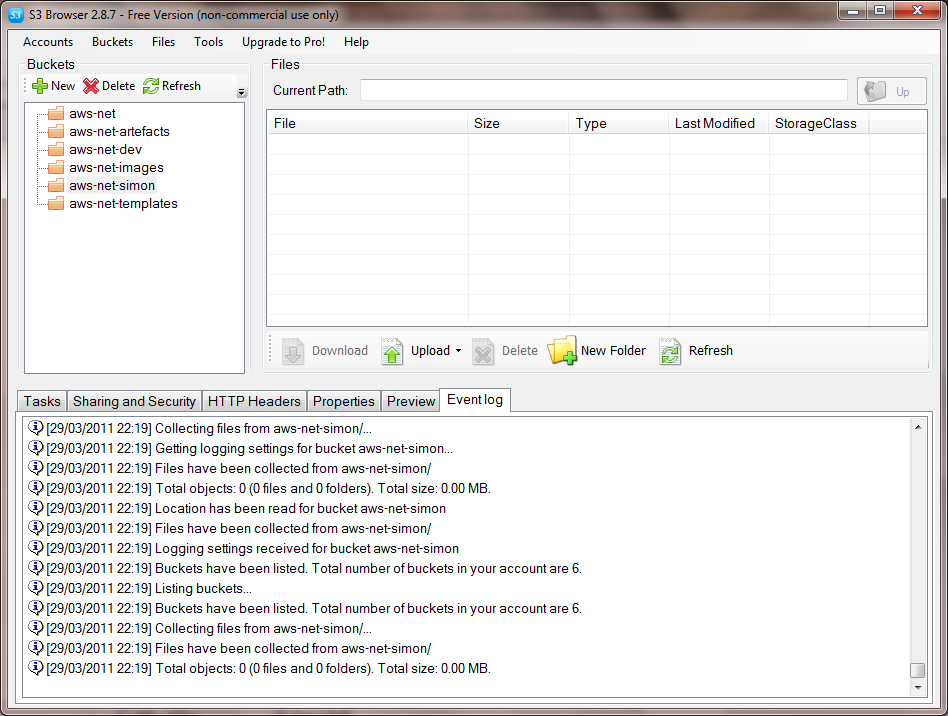
This policy means all Simon can do is create and delete buckets. So when Simon is the user using S3 browser there is no list action allowed.

Logging on as Simon and selecting the new bucket button allows him to create a bucket which he cannot list. If you look closely at the screen shot you can see the EventLog entries showing where he gets access denied when trying to list the bucket but he is able to successfully create the bucket called aws-net-simon



003-58

When you log on as the AWS account owner you can see that all buckets are listed including the new one created by Simon and there are no errors in the Event Log

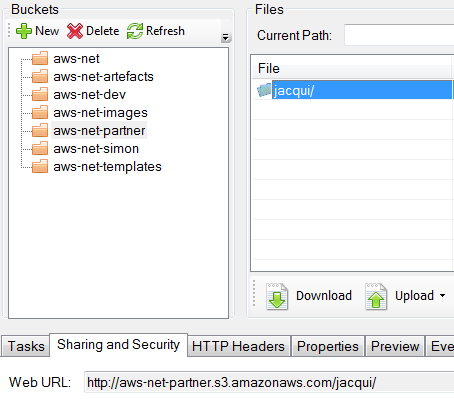


003-59

When accessing S3 using the SDK you need to authenticate using the AWS access key ID and the AWS Secret Access Key which means we can use IAM users to control access at the application level.

We have an application where partners upload objects to S3 and we want to make sure there is no way that one user can actually delete buckets or inadvertently affect other partners data. To do this we create a bucket for partner use and create a folder for each partner in this bucket.

So the bucket looks like this:



003-60

We then associate a bucket policy with the Partner IAM users so in our case for Jacqui we associate the following policy.

|  |
| --- |
| {  "Statement": [  {  "Effect": "Allow",  "Action": "s3:PutObject",  "Resource": "arn:aws:s3:::aws-net-partner/jacqui/\*",  "Condition": {}  }  ]  } |

This policy restricts Jacqui to being allowed to upload data to the URL:

http://aws-net-partner.s3.amazonaws.com/jacqui/

**Bucket Policies and ACL’s**

Bucket policies can be used without IAM accounts in a variety of ways for example they can be used to stop hot linking to your s3 assets and to restrict access to known IP addresses ( never an effective method of implementing security by itself due to IP spoofing).

The bucket policy below restricts access to objects in the bucket aws-net-demo from the indicated websites

|  |
| --- |
| {  "Version":"2008-10-17",  "Id":"http referer policy example",  "Statement":[  {  "Sid":"Allow get requests referred by cloudcomments.com and mysite.com",  "Effect":"Allow",  "Principal":"\*",  "Action":"s3:GetObject",  "Resource":"arn:aws:s3:::aws-net-demo/\*",  "Condition":{  "StringLike":{  "aws:Referer":[  " http://cloudcomments.net/\*",  " http://devopscloud.net/\*"  ]  }  }  }  ]  } |

We have had an initial look at ACL’s when we walked through the permissions tab . ACLs are used to grant access permissions to buckets or objects. When a bucket or object is created S3 creates a default ACL that grants the resource owner full control over the resource.

When we looked at the permission tab we saw hat ACL’s are associated with grantees

Everyone – Allows anyone to access the resource. The requests can be signed (authenticated) or unsigned (anonymous)

Authenticated Users - Allows any Amazon AWS account to access the resource as long as they are authenticated.

Log Delivery – This allows the writing of server access logs to the bucket (as described above)

ACL’s are restricted for use with the above groups and specified AWS accounts (using the canonical User ID) . At the time of writing they could not be used with IAM accounts.

S3 supports a set of predefined ACLs, known as canned ACLs. Each canned ACL has a predefined a set of grantees and permissions. The table below from the S3 documentation lists the canned ACL’s available.

| **Canned ACL** | **Applies to** | **Permissions added to ACL** |
| --- | --- | --- |
| private | Bucket and object | Owner gets FULL\_CONTROL. No one else has access rights (default). |
| public-read | Bucket and object | Owner gets FULL\_CONTROL. The AllUsers group ( see [Specifying a Grantee](http://docs.amazonwebservices.com/AmazonS3/latest/dev/ACLOverview.html#SpecifyingGrantee)) gets the READ access. |
| public-read-write | Bucket and object | Owner gets FULL\_CONTROL. The AllUsers group gets READ and WRITE access. Granting this on a bucket is generally not recommended. |
| authenticated-read | Bucket and object | Owner gets FULL\_CONTROL. The AuthenticatedUsers group gets READ access. |
| bucket-owner-read | Object | Object owner gets FULL\_CONTROL. Bucket owner gets READ access. If specified when creating a bucket, Amazon S3 ignores it. |
| bucket-owner-full-control | Object | Both the object owner and the bucket owner get FULL\_CONTROL over the object. If you specify this canned ACL when creating a bucket, Amazon S3 ignores it. |
| log-delivery-write | Bucket | The LogDelivery group gets WRITE and the READ\_ACP permission on the bucket. For more information on logs, see ([Server Access Logging](http://docs.amazonwebservices.com/AmazonS3/latest/dev/ServerLogs.html)). |

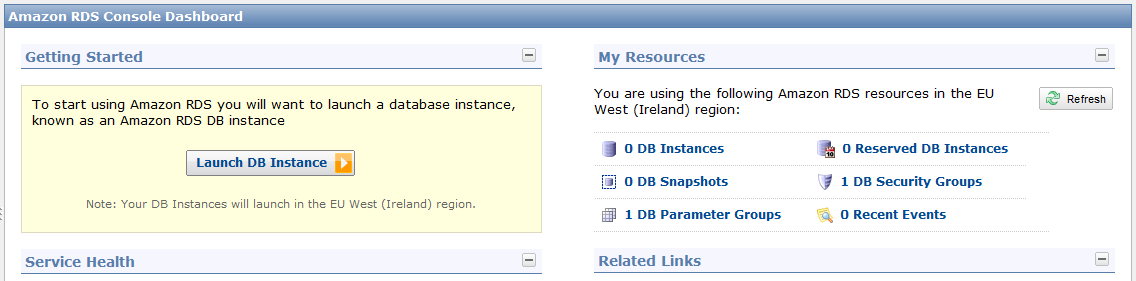
It is probable that you will use both policies and ACL’s to manage your S3 buckets and objects. It is worth remembering that Policies allow or deny certain actions while ACLs grant certain permissions. The actions allowed or denied by policies are actually a superset of the permissions that you can grant by ACLs.

Setting up RDS

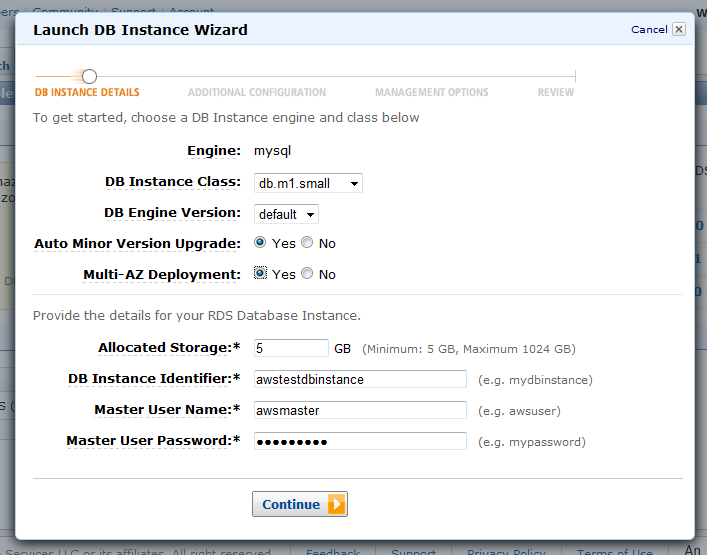
To set up RDS you can as with the majority of AWS services use the console or the Command line.

To set up an instance using the console log on and select the RDS tab

Click the Launch DB Instance button

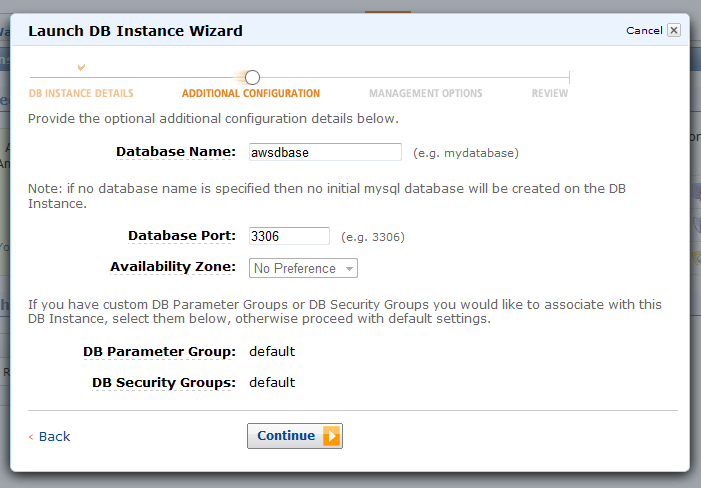
 003-15

This then gives you the initial configuration screen for the wizard that allows you to complete the configuration details for your dbinstance. From the initial page you select your dbinstance, engine type whether it is a multi-AZ deployment and the storage size.



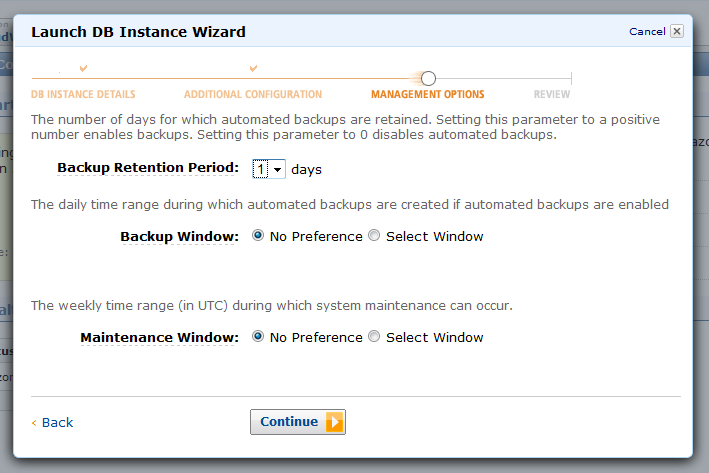
003-16

Select continue to move onto the next screen in the wizard



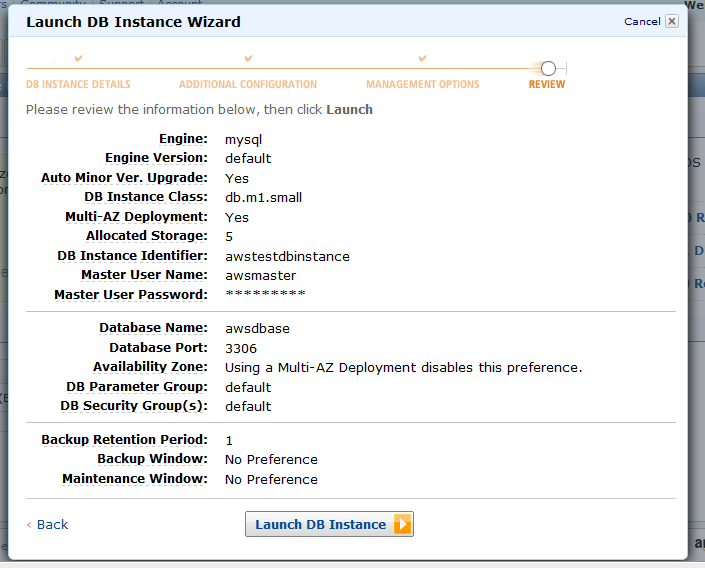
003-17

The third screen is where you fill in the backup options



003-18

You are given the opportunity to then review your options

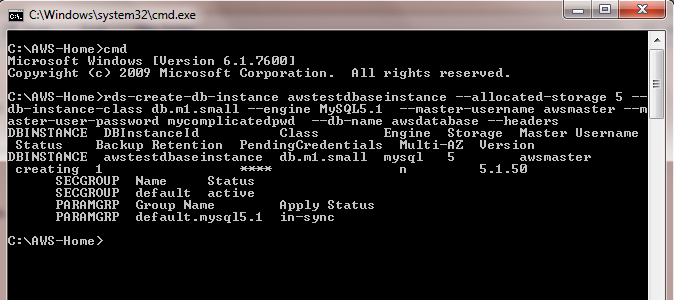


003-19

To create an RDS instance via the command line you need to set up your AWS shell ensuring you have the RDS Command tools set up as described in the section on setting up you AWS shell.

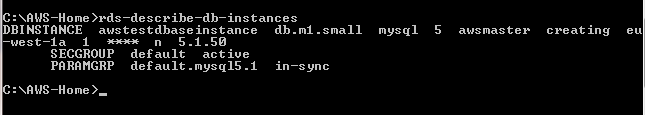
After starting the shell the first thing to do is to create the RDS instance and optionally at the same time your initial database. This is achieved by using the rds-create-db-instance command.

rds-create-db-instance awstestdbaseinstance --allocated-storage 5 --db-instance-class db.m1.small --engine MySQL5.1 --master-username awsmaster --master-user-password mycomplicatedpwd --db-name awsdatabase –headers



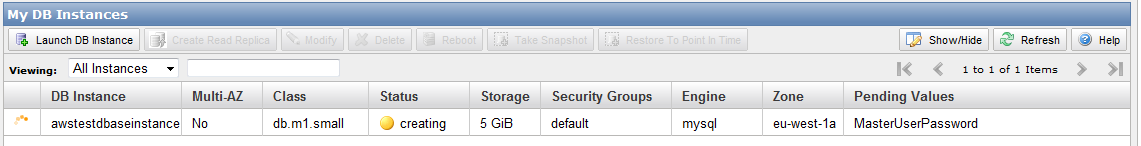
003-20

Progress can be checked by using the rds-db-describe-instances command



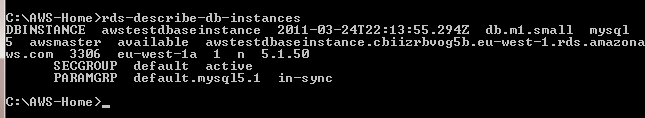
003-21

Or by using the console



003-22

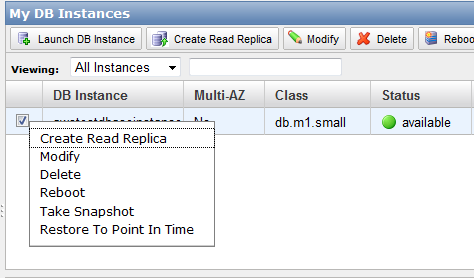
When the database instance is available you can start using it immediately



003-23

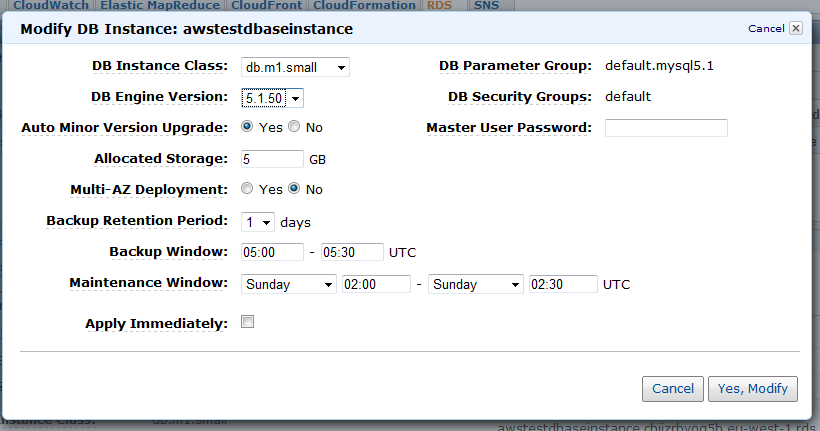
To gain access to your RDS instance you need to make sure that appropriate CDIR and security access is set up using the rds-authorize-db-security-group-ingress command.

You can then carry out actions like creating read replicas, taking snapshots or make other modifications.



003-24

Selecting Modify allows you to change quite a few parameters.



003-25

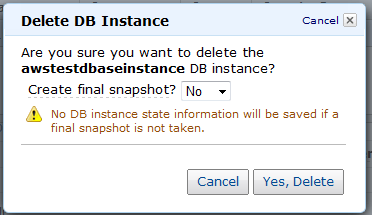
The changes that can be made via the console can also be made using the RDS command line tools.

RDS is just a managed MYSQL instance so you can use it like you would any other MySQL instance assuming you have the appropriate security groups and CDIR set up.

Databases can be created using the mysql command line tool or SQL workbench so to create a second database on my instance I would use a command like the below from the MYSQL command line prompt

-hawstestdbaseinstance.cbiizrbvog5b.eu-west-1.rds.amazonaws.com -P 3306 -uawsmaster –pmycomplicatedpwd < smyseconddatabase.sql

Deleting an RDS instance is a simple case of selecting delete from the console or using the equivalent RDS command line . You can take the opportunity to take a final snapshot ( well you never know)



003-26

Alternative storage options to S3& RDS

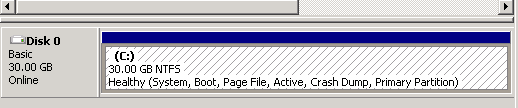
In addition to S3 & RDS AWS also has the following storage options<TODO EDIT Snippets below>

EBS Instance Ephemeral storage, EBS, SimpleDB

Ephemeral storage

Amazon EC2 local instance store volumes (ephemeral drives) provide temporary block-level storage for Amazon EC2 instances. When an Amazon EC2 instance is created from an AMI, in most cases it comes with a preconfigured block of pre-attached disk storage. Unlike EBS volumes, data on instance store volumes persists only during the life of the associated Amazon EC2 instance. The amount of this disk storage ranges from 160 GB up to 1.7 TB, and varies by Amazon EC2 instance type. Larger Amazon EC2 instances have more and larger instance store volumes. Do note that micro instances have no ephemeral storage available. Although the storage space available as ephemeral storage appears generous and can be useful, it is temporary and best used like a scratch volume or RAM disk.

By default a windows 2008 or 2008 R2 instance has 30 GB of space.



003-97

If you start a large instance ( which you are likely to do as minimum to get optimal performance ) then you can get access to extra disk space 850 GB to be exact.

<http://aws.amazon.com/ec2/instance-types/> lists the instance types and also indicates how much instance storage is available this link <http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/index.html?instance-storage-concepts.html> indicates what block device mapping can be used to expose the instance storage

First the storage needs to be attached as a block device. On a windows instance this can be done via the CLI and CloudFormation <Do walkthrough of mounting & exposing within windows>

< ec2-run-instances -b '/dev/sdb=ephemeral0' -b '/dev/sdc=ephemeral1'>

< use for ref I need to describe what is happening <http://www.serveridol.com/2010/08/30/enabling-instance-storage-on-amazon-windows-2008-ami/> >

<Describe how to expose to windows>

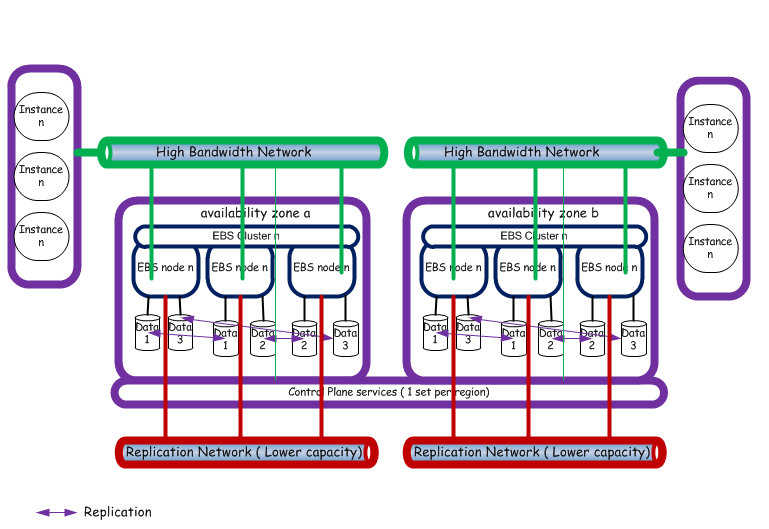
<Talk about use of EBS>

Amazon Elastic Block Store (EBS) Volumes provide durable block-level storage for use with Amazon EC2 instances (virtual machines). Amazon EBS volumes are off-instance, network-attached storage that persists independently from the running life of a single Amazon EC2 instance. After an EBS volume has been attached to an Amazon EC2 instance the instance can interact with it just as if it would a physical hard disk drive, typically by formatting it with an applicable file system. An EBS volume can be used to boot an Amazon EC2 instance (EBS AMIs only), and attach multiple EBS volumes to a single Amazon EC2 instance. Note, however, that any single EBS volume may be attached to only one Amazon EC2 instance at any point in time

Amazon EBS volumes are designed to be highly available. However, because EBS volumes are created in a particular Availability Zone, they will be unavailable if the Availability Zone itself is unavailable. Note that while any single EBS volume is constrained to single Availability Zone, an EBS snapshot of a volume is available across all the Availability Zones within a Region, and you can use an EBS snapshot to create one or more new EBS volumes in any Availability Zone

The durability and availability is attained because the EBS infrastructure automatically replicates EBS volume data between EBS nodes within a region. EBS Nodes Communicate with other  EBS nodes, with EC2 instances, and with the EBS control plane services via a high bandwidth network. The control plane service accepts user requests and propagates them to the appropriate EBS cluster. An EBS Cluster manages a set of EBS Nodes

The diagram below illustrates the EBS infrastructure



003-96

To use EBS volumes you need to attach an EBS volume and then mount it from within the instance.

Attaching an EBS volume can be achieved via the CLI, via the console, CloudFormation or the API tools

<describe attaching an EBS volume – its the same for linux & windows apart from the mount point>

< Then describe how to expose to windows>

<SimpleDB>

SimpleDB is a highly available, scalable, and flexible non-relational data store that offloads much of the work of database administration and associated systems management. SimpleDB, has a “schema-less” data model, and can store data items comprised of a flexible number of name/value pairs

<Talk about setting up SimpleDB>

Setting up SQS

Amazon Simple Queue Service (Amazon SQS) is a reliable, highly scalable, hosted queue for storing messages as they travel between computers.  SQS is a vital cog in architecting for the Cloud as it provides a way to decouple components that constitute an application.

Key features of SQS are

* The message body can contain up to 64 KB of text in any format (default is 8KB).
* Messages can be retained in queues for up to 14 days (default is 4 days).
* Messages can be sent and read simultaneously.
* When a message is received, it becomes “locked” while being processed. This keeps other computers from processing the message simultaneously. If the message processing fails, the lock will expire and the message will be available again. In the case where the application needs more time for processing, the “lock” timeout can be changed dynamically via the ChangeMessageVisibility operation.

Queues are created via the API or via CloudFormation. There is a Javascript scratchpad available at <http://docs.amazonwebservices.com/AWSSimpleQueueService/latest/SQSGettingStartedGuide/> to help you work with SQS .

<TODO creating SQS via the scratchpad & via cloudformation>

Can also be created via the api

Setting up SES

Sending email from Amazon used to be a hit or miss affair due to issues with problems associated with black lists and spam originating from email sent from Amazon routable IP addresses. To reliably send email from AWS would require setting up SMTP relays whether to your own hosted SMTP gateways or to a 3rd party SMTP service.

AWS introduced SES to simplify the implementation of sending email from AWS and to overcome some of the issues.

On first signing up with AWS SES you will be granted access to a sandbox environment that only allows you to send email to addresses that you have verified and also has sending limits applied. Before you can send an email though the recipient email address needs to be verified this includes your own email address. Amazon supply perl scripts to help get you started. So you either need to get Perl installed so you can use the script or we recommend using the AWS SDK to write a small .NET console application to do the verification step.

The salient bit of verification code may look similar to this:

|  |
| --- |
| using Amazon.SimpleEmail;  using Amazon.SimpleEmail.Model;  static class Program {  static void Main() {  var client = new AmazonSimpleEmailServiceClient(“AWSAccessKey", "AWSSecretKey");  var request = new VerifyEmailAddressRequest { EmailAddress = "EmailAddresstoVerify" };  client.VerifyEmailAddress(request);  }  } |

Once verified you can then send email from AWS using code similar to that below to allow the sending of emails to the recipient email address :

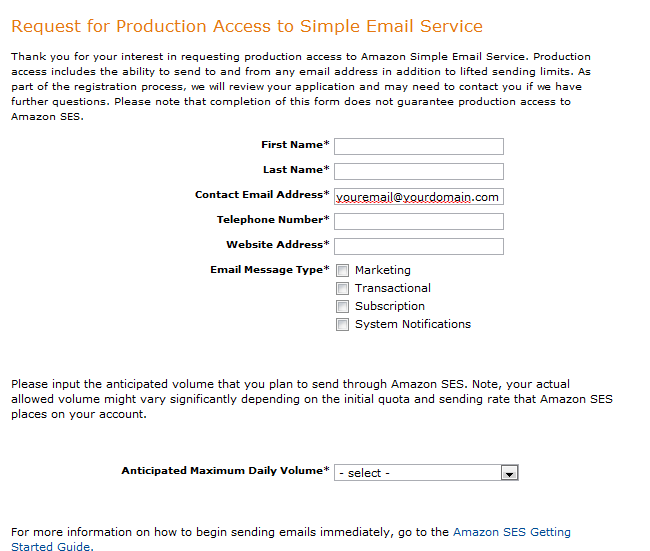
|  |
| --- |
| using Amazon.SimpleEmail;  using Amazon.SimpleEmail.Model;  private void SendEmail(string FromAddress, List ToAddresses, string Subject, string mailBody)  {  var client = new AmazonSimpleEmailServiceClient("AWSAccessKey", "AWSSecretKey");  SendEmailRequest emailreq = new SendEmailRequest()  .WithDestination(new Destination() { BccAddresses = ToAddresses })  .WithSource(FromAddress)  .WithReturnPath(FromAddress)  .WithMessage(  new Amazon.SimpleEmail.Model.Message(new Content(Subject),  new Body().WithText(new Content(mailBody))));  var response = client.SendEmail(emailreq); |

The verification step will send an email to the email address you wish to allow email to be sent to. This email will contain a link that will need to be clicked to verify the email address.

One cannot go into production using the sandbox so further configuration steps are required to use SES in production mode. This two step process is an effort from AWS to try and combat the abuses of the service.

To request production mode access that will do away with the requirement of verifying email addresses the first thing that needs doing is to fill in the form at :

<https://aws-portal.amazon.com/gp/aws/html-forms-controller/contactus/SESAccessRequest>



003-95

Once granted production access the restrictions on To: or Reply-To: addresses are lifted but any additional From: addresses need to be verified.

To use SES effectively and reliably one needs to implement SMTP authentication. Implementing authentication provides a means for ISP s to validate that the email is originating from an authorised sender. To do this you will need to set up one or more of the following methods for setting up authentication when using SES:

Sender Policy Framework (SPF): An email sender publishes one or more DNS records that establish the sending domain's identity. The DNS records identify the hosts that are authorized to send email. ISPs can authenticate a host by comparing its IP address with the set of IP addresses specified in the SPF record. For the sending domain.

If the from domain already has an SPF record then you need to add the following:

include:amazonses.com

If no SPF record exists then you will need to add the following text record:

v=spf1 include:amazonses.com ?all

Sender ID: An email sender publishes one or more DNS records to establish the sending domain's identity. An ISP can authenticate a host by using an algorithm known as Purported Responsible Address (PRA). The PRA algorithm tries to find the email address of the responsible party for the message, out of several From:addresses that could be present in the header and body. When the address is determined, the ISP can authenticate the host by comparing its IP address with the set of IP addresses specified in the Sender ID record of the domain of the purported responsible address.

If the "From" domain already has a Sender ID record, then you need to add the following mechanism to it:

include:amazonses.com

If the From domain does not have a Sender ID record then the following text record should be added to the DNS records for the domain:

spf2.0/pra include:amazonses.com ?all

It is strongly recommend that to ensure optimal delivery rates and to help prevent spoofing and phishing, SES users maintain both SPF records (v=spf1) and Sender ID records (spf2.0/pra) in their DNS servers

DKIM can also be used to sign emails

There are limits on the usage of SES which are determined by SES and the way these are applied are outlined in the planning ahead section of the SES developers guide: <http://docs.amazonwebservices.com/ses/latest/DeveloperGuide/>

It is worth reading through this to understand how it works .

Setting up Simple Notification Services

Simple Notification Services (SNS) is a web service that is used to set up, operate, and send notifications from  AWS. When using SNS a topic needs to be created. A topic is a communication channel to send messages and subscribe to notifications. It provides an access point for publishers and subscribers to communicate with each other. SNS follows the “publish-subscribe” (pub-sub) messaging paradigm, with notifications being delivered to clients using a “push” mechanism that eliminates the need to periodically check or “poll” for new information and updates

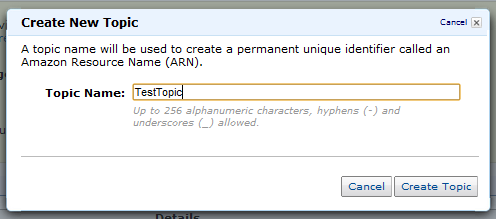
SNS is particularly useful when associated with alarms. SNS topics can be created via the console, the command line, via API calls and by using CloudFormation.

SNS Endpoints can be a URL ( via a http post), email addresses ( text or JSON object) , or the Amazon Resource Name (ARN) of SQS queues .

To create an SNS topic via the console:

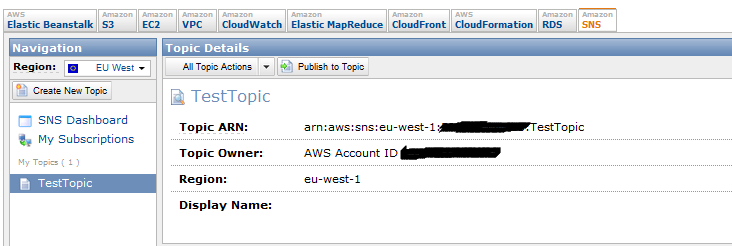
From the SNS tab click on the Create new topic button which starts the Create Topic wizard

Enter the name of the Topic



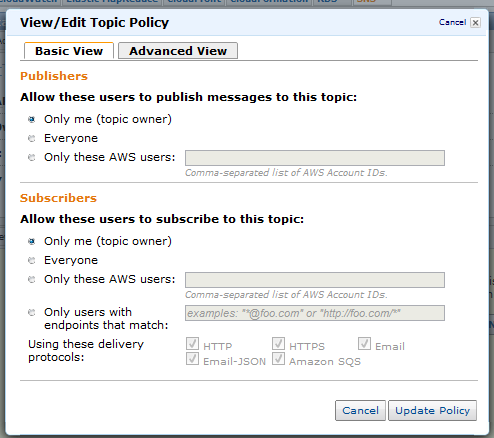
003-90

Click the create Topic button. The topic appears in the navigation bar



003-91

From the All Topic Action selecting the View/Edit Topic policy gives access to the topic policy where access can be granted to publishers and subscribers. IAM policies can also be used to manage access to SNS topics.



003-92

Using CloudFormation to create an SNS topic:

To set up an SNS topic you have to create an SNS Topic resource

|  |
| --- |
| "TestTopic" : {  "Type" : "AWS::SNS::Topic",  "Properties" : {  "Subscription" : [ {  "Endpoint" : { "Ref" : "SupportEmail" },  "Protocol" : "email"  } ]  }  } |

The topic can then be associated with alarm resources as the alarm action

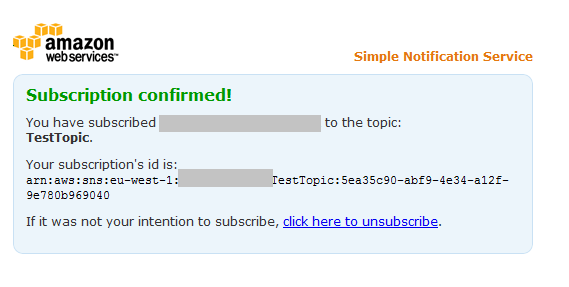
|  |
| --- |
| "AlarmActions" : [ { "Ref" : "TestTopic" } ], |

Subscribing to a topic:

If the end point is an email then a confirmation of the subscription will need to be acknowledged via the email sent to the subscribing email

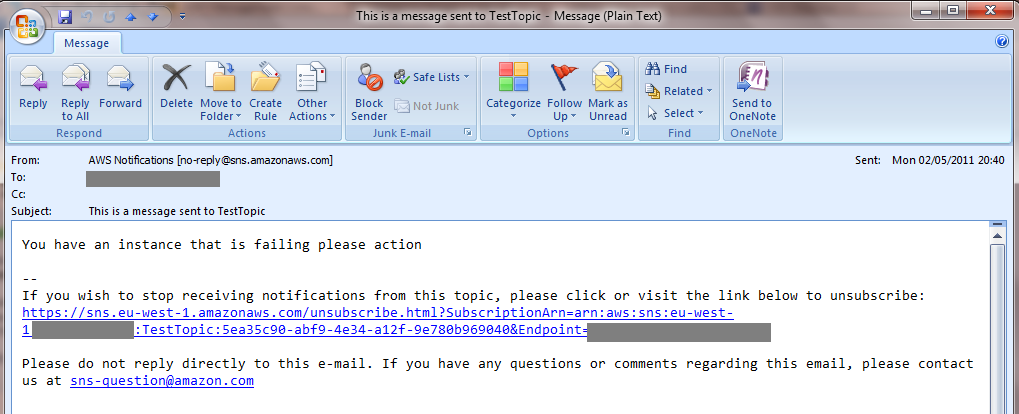
|  |
| --- |
| You have chosen to subscribe to the topic:  **arn:aws:sns:eu-west-1:XXXXXXXXX:TestTopic**  To confirm this subscription, click or visit the link below (If this was in error no action is necesary):  Confirm subscription  Please do not reply directly to this e-mail. If you wish to remove yourself from recieving all future SNS subscription confirmation requests please send email to sns-opt-out |

Clicking on the confirmation link will redirect to a confirmation page



003-93

After confirming any messages sent to SNS will be sent to the email end point.



003-94

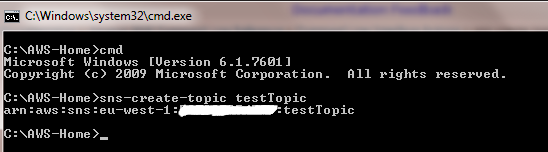
The command line provides access to all the features you have accessible from the console the reference docs can be found at <http://docs.amazonwebservices.com/sns/latest/cli/>

The SNS command line tools are simple to use (we assume you have downloaded the CLI tools and have set them up as described in the setting up an AWS Shell environment)

To create a topic using the CLI:

sns-create-topic TopicName

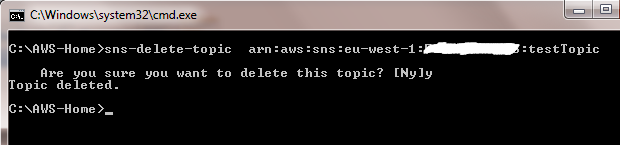
Which returns the topic arn



003-88

To delete a topic you need to pass the sns-delete-topic the arn of the topic

sns-delete-topic arn-Of-Topic



003-89

DNS configuration when using AWS services

On configuring AWS resources that are typically granted access publically they are configured with DNS names that reflect the aws domain .

For an s3 bucket it will be of the format:

**Yourbucketname.s3.amazonaws.com**

For an ELB it will be of the format:

**YourLoadBalancername-GUID-availabiltyzone.elb.amazonaws.com**

For CloudFront it will be of the format:

**GUID.cloudfront.net**

These DNS names are unique but do not reflect the domain that confers an identity as part of an application deployment.

To allow the use of a DNS record that makes use of a domain you own and can make DNS changes for a CNAME record needs to be configured for the resource. Note that you cannot use a root domain when working with CNAME records and will have to use sub-domains. For example you cannot use youydomain.com but will need to use a sub-domain of the format subdomain.yourdomain.com

To configure a CNAME record for an S3 bucket .

1. Name the bucket after the sub-domain you want to map to the bucket

For example if you want to map images.yourdomain.com to a bucket the bucket should be named images.yourdomain.com which would resolve as images.yourdomain.s3.amazonaws.com

1. Create a CNAME record that maps to the bucket.

For the domain yourdomain.com the CNAME entry for the images bucket would have the following values:

|  |  |  |
| --- | --- | --- |
| Name | Type | Value |
| images | CNAME | images.yourdomain.s3.amazonaws.com. |

To configure a CNAME record for an ELB

1. Create a CNAME record for the domain that maps to the ELB DNS name

For the domain yourdomain.com the CNAME entry to map www.yourdomain.com to the AWS DNS name for your Elastic load balancer would have the following values:

|  |  |  |
| --- | --- | --- |
| Name | Type | Value |
| www | CNAME | YourLoadBalancername-GUID-availabiltyzone.elb.amazonaws.com |

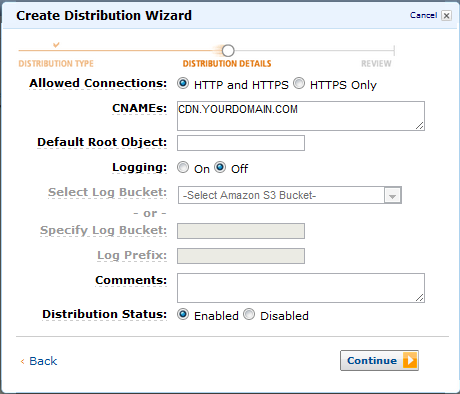
To configure a CNAME record for CloudFront

This is similar to setting up a CNAME record For an ELB

1. Create a CNAME record for the domain that maps to the CloudFront DNS name. For the domain yourdomain.com the CNAME entry to map CDN.yourdomain.com to the AWS DNS name for your CloudFront distribution would have the following values

|  |  |  |
| --- | --- | --- |
| Name | Type | Value |
| CDN | CNAME | GUID.cloudfront.net |

1. Update the CloudFront Distribution configuration with the CNAME alias configured in step . This can be done via the API or the AWS Console



003-87

Amazon Route 53

AWS also have a service called Route 53 which is a DNS service. If you are using a new Domain that has no dependencies on services outside of AWS it may be prudent to investigate this as an alternative way to manage Domains used with AWS resources. Note this is NOT a domain registration service it To allow route 53 to respond to DNS queries for your domain do this you need to register the Amazon route 53 hosted zone name service with the registrar.

Anatomy of an AWS Template

Before you start writing a Cloudformation template or modifying an existing template you need to understand the components that go towards creating one. There are 6 basic components that can be used in the JSON Cloudformation templates that you need to become familiar with.

Format version (optional): The AWS CloudFormation template version against which the template was written

Description (optional): JSON text string description of template or part of template

Parameters (optional): String or comma separated list. The values can be overridden at run time. Parameters are dereferenced in resources and outputs section of template e.g you can declare a Parameter called InstanceType with a default of t1.micro that can be overridden at Stack instantiation to be an alternative InstanceType.

|  |
| --- |
| "InstanceType" : {  "Type" : "String",  "Default" : "t1.micro",  "Description" : " 't1.micro' | 'm1.large' | 'c1.xlarge' "  }, |

Mappings (optional): Allow the passing of conditional parameter values used with the function Fn::FindInMap. It is similar to a case statement in usage. Used in conjunction with Parameters. Each mapping has a unique name in a template and consists of a number of key-attribute pairs. Each Attribute is a literal string

|  |
| --- |
| "Mappings" : {  "RegionMap" : {  "us-east-1" : {  "AMI" : "ami-8e1fece7",  "AvailabilityZone" : "us-east-1b"  },    "eu-west-1" : {  "AMI" : "ami-45cefa31",  "AvailabilityZone" : "eu-west-1a"  }    }  }, |

When using the function Fn:FindInMap it needs to be passed the Map name, the Reference key and the return value

Resources: AWS resources such as instances, RDS etc which are declared as members of the AWS stack . Resources declared in the Resources section contain a Properties section, in which you declare both required and optional properties for the resource.

|  |
| --- |
| "Resources" : {  "Ec2Instance" : {  "Type" : "AWS::EC2::Instance",  "Properties" : {  "KeyName" : { "Ref" : "KeyName" },  "AvailabilityZone" : { "Fn::FindInMap" : [ "RegionMap", { "Ref" : "AWS::Region" }, "AvailabilityZone" ]},  "ImageId" : { "Fn::FindInMap" : [ "RegionMap", { "Ref" : "AWS::Region" }, "AMI" ]},  "InstanceType" : { "Ref" : "InstanceType"}  }  }  }, |

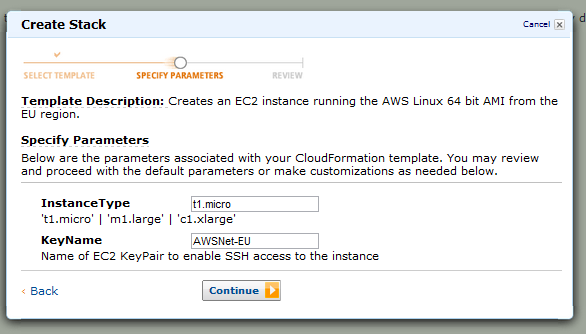
Outputs (optional): Messages that can be returned as part of the cfn-describe-stacks command

|  |
| --- |
| "PublicIP" : {  "Description" : "Public IP address of the newly created EC2 instance",  "Value" : { "Fn::GetAtt" : [ "Ec2Instance", "PublicIp" ] }  } |

So putting the above together to create a simple AWS stack that creates a single EC2 instance :

|  |
| --- |
| {  "AWSTemplateFormatVersion" : "2010-09-09",  "Description" : "Creates an EC2 instance running the AWS Linux 64 bit AMI from the EU region. ",    "Parameters" : {  "KeyName" : {  "Description" : "Name of EC2 KeyPair to enable SSH access to the instance",  "Default" : "AWSNet-EU",  "Type" : "String"  },  "InstanceType" : {  "Type" : "String",  "Default" : "t1.micro",  "Description" : " 't1.micro' | 'm1.large' | 'c1.xlarge' "  }      },  "Mappings" : {  "RegionMap" : {  "us-east-1" : {  "AMI" : "ami-8e1fece7",  "AvailabilityZone" : "us-east-1b"  },    "eu-west-1" : {  "AMI" : "ami-45cefa31",  "AvailabilityZone" : "eu-west-1a"  }    }  },    "Resources" : {  "Ec2Instance" : {  "Type" : "AWS::EC2::Instance",  "Properties" : {  "KeyName" : { "Ref" : "KeyName" },  "AvailabilityZone" : { "Fn::FindInMap" : [ "RegionMap", { "Ref" : "AWS::Region" }, "AvailabilityZone" ]},  "ImageId" : { "Fn::FindInMap" : [ "RegionMap", { "Ref" : "AWS::Region" }, "AMI" ]},  "InstanceType" : { "Ref" : "InstanceType"}  }  }  },    "Outputs" : {  "InstanceId" : {  "Description" : "InstanceId of the newly created EC2 instance",  "Value" : { "Ref" : "Ec2Instance" }  },  "AZ" : {  "Description" : "Availability Zone of the newly created EC2 instance",  "Value" : { "Fn::GetAtt" : [ "Ec2Instance", "AvailabilityZone" ] }  },  "PublicIP" : {  "Description" : "Public IP address of the newly created EC2 instance",  "Value" : { "Fn::GetAtt" : [ "Ec2Instance", "PublicIp" ] }  }  }  } |

You can now upload the template via the console or preferably place it in your template bucket. If you upload a file AWS creates a template bucket for you and you can subsequently use the template from there. If you opt to upload the template from the file each time then you will get a copy of the template copied to this bucket



003-78

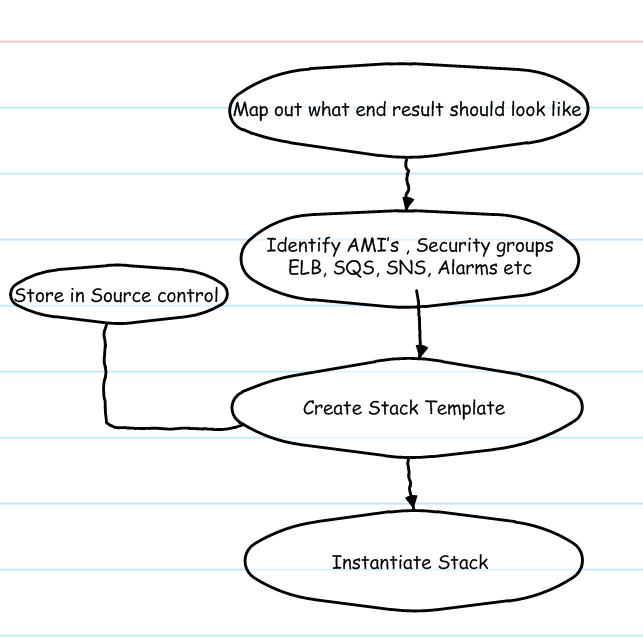
You can check whether your template is valid by using the cfn-validate-template --template-file command or even by attempting to create a stack .

You can find template help from here:

<http://docs.amazonwebservices.com/AWSCloudFormation/latest/UserGuide/index.html?template-guide.html>

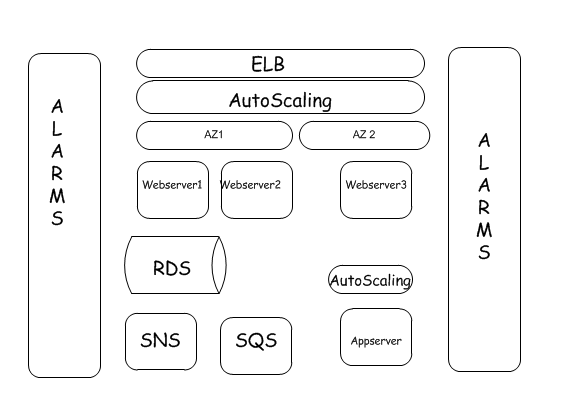
That was a trivial example just to get you used to writing CloudFormation templates now we will go through the process required to create a more typical AWS CloudFormation Template.

The sketch below illustrates the process that needs to be followed when creating an AWS template.



003-79

The diagram below outlines a more realistic AWS configuration that we would want our template to describe



003-80

Now that we know what the target stack looks like the next thing to do is to list out the resources required, define the security group rules, define the alarm thresholds etc. Typically as part of the stack creation you would create everything from scratch as an AWS stack is the starting point for your AWS estate.

AS the stack gets more complicated it is easy to miss out a comma or get the syntax incorrect in some way. As you add more resources check the syntax using cfn-validate-template at opportune moments and take the opportunity to spin up the stack and then tear it down to make sure it’s doing what you expect it to.

Using the diagram above the resultant components we need to set up looks like this when represented by an AWS stack template:

|  |
| --- |
| {  "AWSTemplateFormatVersion" : "2010-09-09",  "Description" : "Creates a more realistic Stack as an example - web server config multi AZ with alarm & notification topic ",    "Parameters" : {  "KeyName" : {  "Description" : "Name of EC2 KeyPair to enable SSH access to the instance",  "Default" : "AWSNet-EU",  "Type" : "String"  },  "InstanceType" : {  "Type" : "String",  "Default" : "t1.micro",  "Description" : " 't1.micro' | 'm1.large' | 'c1.xlarge' "  },    "WebServerPort" : {  "Description" : "TCP/IP port of the web server",  "Type" : "String",  "Default" : "80"  },    "WebRole" : {  "Description" : "Chef Role that will be applied to the webserver Instances",  "Type" : "String",  "Default" : "WebBase"  },    "OperatorEmail" : {  "Default" : "yourOperator@yourcompany.com",  "Description" : "EMail address to notify if there are any operational issues",  "Type" : "String"  }      },  "Mappings" : {  "RegionMap" : {  "us-east-1" : {  "AMI" : "ami-8e1fece7"    },    "eu-west-1" : {  "AMI" : "ami-45cefa31"    }    }  },    "Resources" : {    "WebServerGroup" : {  "Type" : "AWS::AutoScaling::AutoScalingGroup",  "Properties" : {  "AvailabilityZones" : { "Fn::GetAZs" : ""},  "LaunchConfigurationName" : { "Ref" : "WebLaunchConfig" },  "MinSize" : "1",  "MaxSize" : "3",  "LoadBalancerNames" : [ { "Ref" : "WebLoadBalancer" } ]  }  },    "WebLaunchConfig" : {  "Type" : "AWS::AutoScaling::LaunchConfiguration",  "Properties" : {  "KeyName" : { "Ref" : "KeyName" },  "ImageId" : { "Fn::FindInMap" : [ "RegionMap", { "Ref" : "AWS::Region" }, "AMI" ]},  "UserData" : { "Fn::Base64" : { "Ref" : "WebRole" }},  "SecurityGroups" : [ { "Ref" : "ProductionSecurityGroup" } ],  "InstanceType" : { "Ref" : "InstanceType" }  }  },    "WebLoadBalancer" : {  "Type" : "AWS::ElasticLoadBalancing::LoadBalancer",  "Properties" : {  "AvailabilityZones": { "Fn::GetAZs" :{"Ref": "AWS::Region"}},  "Listeners" : [ {  "LoadBalancerPort" : "80",  "InstancePort" : { "Ref" : "WebServerPort" },  "Protocol" : "HTTP"  } ],  "HealthCheck" : {  "Target" : { "Fn::Join" : [ "", ["HTTP:", { "Ref" : "WebServerPort" }, "/"]]},  "HealthyThreshold" : "3",  "UnhealthyThreshold" : "5",  "Interval" : "30",  "Timeout" : "5"  }  }  },    "AppServerGroup" : {  "Type" : "AWS::AutoScaling::AutoScalingGroup",  "Properties" : {  "AvailabilityZones" : { "Fn::GetAZs" : ""},  "LaunchConfigurationName" : { "Ref" : "AppServerLaunchConfig" },  "MinSize" : "1",  "MaxSize" : "1"  }  },  "AppServerLaunchConfig" : {  "Type" : "AWS::AutoScaling::LaunchConfiguration",  "Properties" : {  "KeyName" : { "Ref" : "KeyName" },  "ImageId" : { "Fn::FindInMap" : [ "RegionMap", { "Ref" : "AWS::Region" }, "AMI" ]},  "UserData" : { "Fn::Base64" : "AppServer" },  "SecurityGroups" : [ { "Ref" : "ProductionSecurityGroup" } ],  "InstanceType" : { "Ref" : "InstanceType" }  }  },    "MessageQueue" : {  "Type" : "AWS::SQS::Queue",  "Properties" : {  "VisibilityTimeout" : "1"  }  },    "ProductionSecurityGroup" : {  "Type" : "AWS::EC2::SecurityGroup",  "Properties" : {  "GroupDescription" : "Enable SSH access and HTTP access on the inbound port ",  "SecurityGroupIngress" : [ {  "IpProtocol" : "tcp",  "FromPort" : "22",  "ToPort" : "22",  "CidrIp" : "0.0.0.0/0"  },  {  "IpProtocol" : "tcp",  "FromPort" : { "Ref" : "WebServerPort" },  "ToPort" : { "Ref" : "WebServerPort" },  "CidrIp" : "0.0.0.0/0"  } ,  {  "IpProtocol" : "tcp",  "FromPort" : "3389",  "ToPort" : "3389",  "CidrIp" : "172.16.30.0/24"  }    ]  }  },    "AlarmTopic" : {  "Type" : "AWS::SNS::Topic",  "Properties" : {  "Subscription" : [ {  "Endpoint" : { "Ref" : "OperatorEmail" },  "Protocol" : "email"  } ]  }  },    "CPUAlarmHigh" : {  "Type" : "AWS::CloudWatch::Alarm",  "Properties" : {  "AlarmDescription" : "Alarm if CPU too high or metric disappears indicating instance is down",  "AlarmActions" : [ { "Ref" : "AlarmTopic" } ],  "InsufficientDataActions" : [ { "Ref" : "AlarmTopic" } ],  "MetricName" : "CPUUtilization",  "Namespace" : "AWS/EC2",  "Statistic" : "Average",  "Period" : "60",  "EvaluationPeriods" : "1",  "Threshold" : "10",  "ComparisonOperator" : "GreaterThanThreshold",  "Dimensions" : [ {  "Name" : "AutoScalingGroupName",  "Value" : { "Ref" : "WebServerGroup" }  } ]  }  },  "TooManyUnhealthyHostsAlarm" : {  "Type" : "AWS::CloudWatch::Alarm",  "Properties" : {  "AlarmDescription" : "Alarm if there are too many unhealthy hosts.",  "AlarmActions" : [ { "Ref" : "AlarmTopic" } ],  "InsufficientDataActions" : [ { "Ref" : "AlarmTopic" } ],  "MetricName" : "UnHealthyHostCount",  "Namespace" : "AWS/ELB",  "Statistic" : "Average",  "Period" : "60",  "EvaluationPeriods" : "1",  "Threshold" : "0",  "ComparisonOperator" : "GreaterThanThreshold",  "Dimensions" : [ {  "Name" : "LoadBalancerName",  "Value" : { "Ref" : "WebLoadBalancer" }  } ]  }  }      },    "Outputs" : {    "URL" : {  "Description" : "URL of the website",  "Value" : { "Fn::Join" : [ "", [ "http://", { "Fn::GetAtt" : [ "WebLoadBalancer", "DNSName" ]}]]}  }      }  } |

It is inadvisable to use CloudFormation to set up resources that will hold your data and storage. At the time of writing this book there was no Stack Protection feature so to prevent accidental deletion of your data stores we would advise creating these outside of the AWS Stack.

Now that we have created a template the next thing to do is to actually use it to create an AWS Stack.

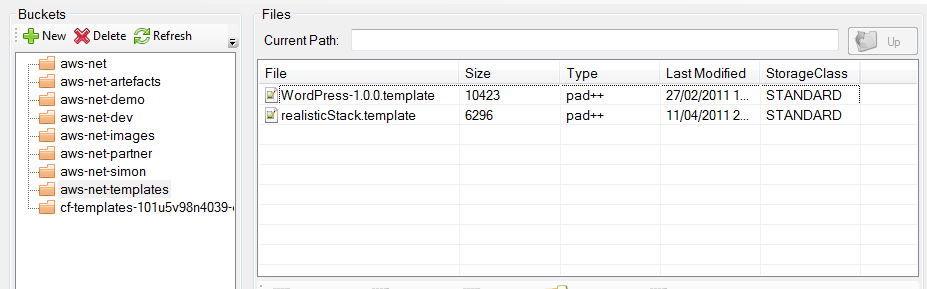
Using CloudFormation

If you have followed through the creation of an AWS Stack template in the previous section you will have seen that the creation of templates is relatively painless one you get used to the resource types and properties available.

The first thing to do is to add CloudFormation to your AWS shell if it hasn’t already been added (See section on setting up your Windows AWS Shell).

Templates are stored in an S3 bucket . We will be using a bucket called aws-net-templates

We have uploaded to the bucket the template we created earlier and also the wordpress sample template.



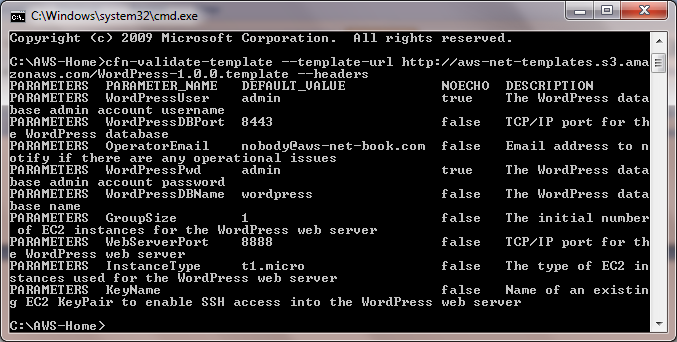
003-01

For the rest of this section on using the AWS template we will use the wordpress template as it is slightly more complicated than the example we walked through earlier and makes use of more resources. We will describe using the command line interface as we showed how to use the AWS console in the section on creating templates.

Once you have created a template verify it using the cfn-validate-template command:

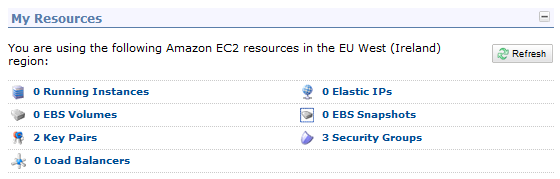
cfn-validate-template --template-url http://aws-net-templates.s3.amazonaws.com/WordPress-1.0.0.template --headers.

This checks the syntax only it will not verify that the AMI values you have are correct you will only find that out when the stack fails to create.



003-02

Before we created the stack there were no resources configured:



003-03



003-04

Create the Stack using the cfn-create-stack command:

cfn-create-stack awsTestStack --template-url https://aws-net-templates.s3.amazonaws.com/WordPress-1.0.0.template --parameters="KeyName=AWSNet-EU"

This returns the unique Stack identifier. This action then creates your stack so expect at least a 10 minute tea break while it does all the lifting and shifting for you. The region will be determined by the region variables you have set up for your AWS shell. .

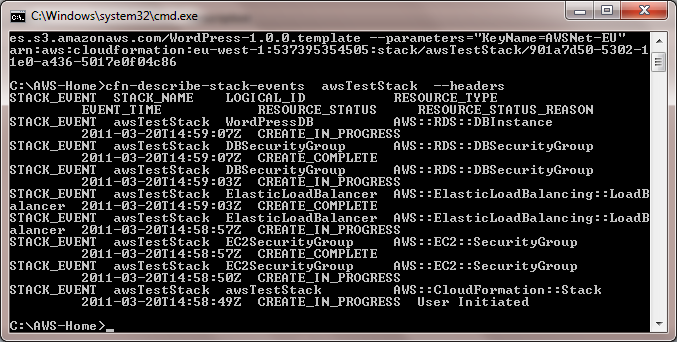


003-05

Use the cfn-describe-stack –events command to monitor progress:

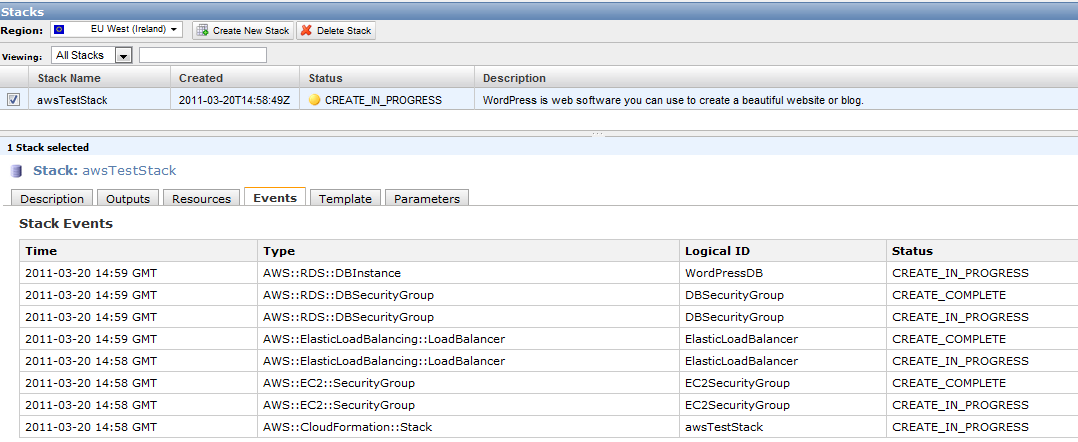
cfn-describe-stack-events awsTestStack --headers

In the example below you can see that it still has a number of components to create



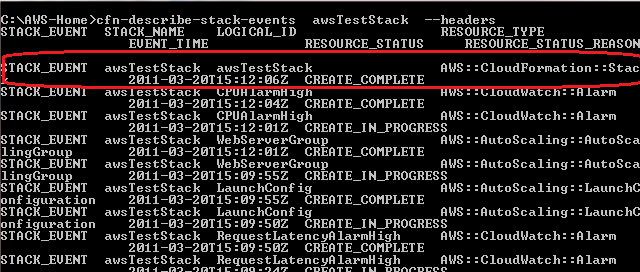
003-06

You can also monitor progress using the AWS Console



003-07

The stack is ready which is shown by the cfn-describe stack indicating that AWS::CloudFormation::Stack is returning a status value of : CREATE\_COMPLETE



003-08

You can also view the status via the AWS Dashboard:



003-09

You can check the status of your Stacks by using the cfn-describe-stacks command:

cfn-describe-stacks awsTestStack

If you do not pass your stack name in it will report back on all your stacks.



003-10

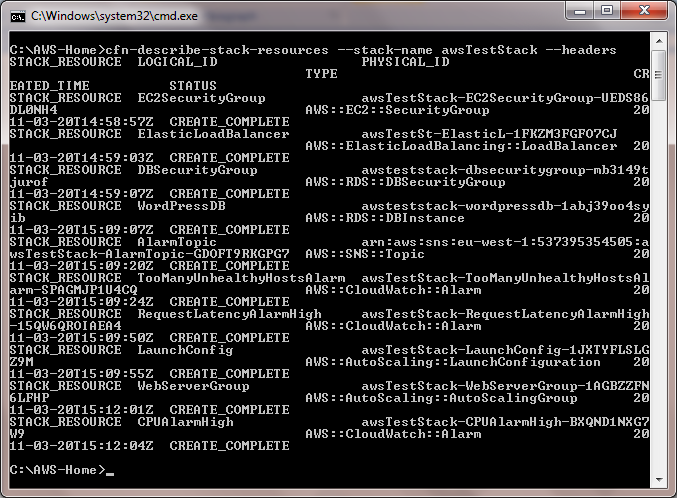
You can see from various AWS Dashboard views the new components that have been created as part of the stack:



003-11

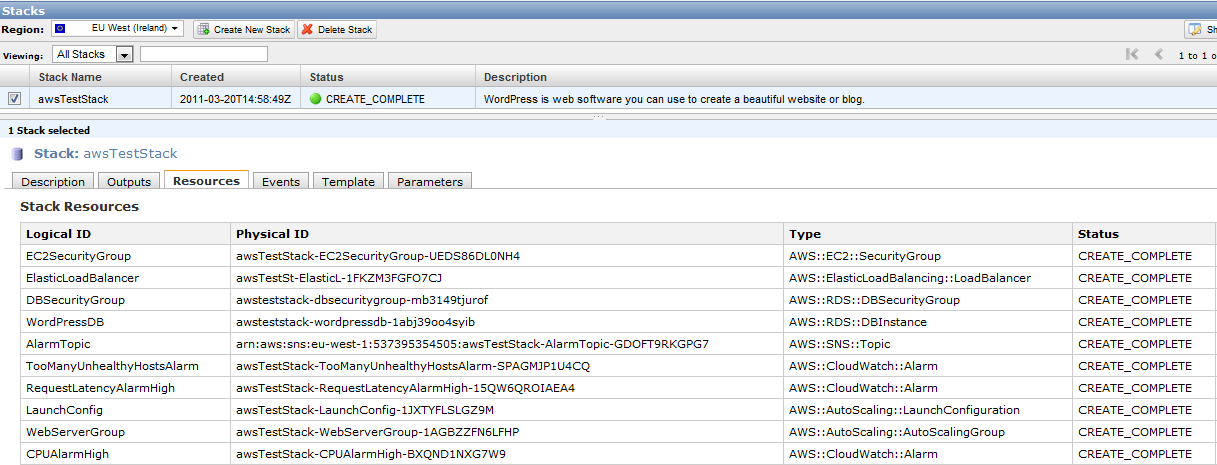
Using the cfn-describe-stack-resources command to view the stacks resources:

cfn-describe-stack-resources --stack-name awsTestStack --headers



003-12

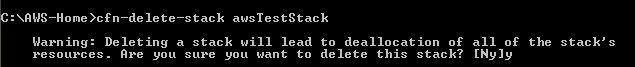
From the CloudFormation Dashboard you can also view the status:



003-13

Deleting the stack is achieved by using the cfn-delete-stack command:

cfn-delete-stack awsTestStack



003-14

Using CloudFormation provides you with a declarative, reproducible way of creating identical stacks. You should add your template files to your source control repository. The use of stacks in running up test and pre-prod environments that exactly mirror the eventual target environment and being able to tear them down when done testing is one of the killer features when using AWS. Not only does it allow you to test a fully representative target it also allows you to manage costs as those test environments are only running when you need them . The use of Stacks makes the management simple.

We envisage a whole community of stack templates being made available to add to the Amazon provided templates.

The recommended approach is to use AWS stacks for all your environments. That however would make for a very short introduction on using AWS and you do need to be familiar with all the tools available.

CloudFormation allows you to provision your AWS resources but you also need to configure that resource which is where a configuration management solution like chef comes into play.

Using Chef

The first thing you need to do is to sign up with the Opscode hosted platform: <http://opscode.com/>. Using the hosted platform gives you the ability to manage 5 nodes free and is a good way to get your hands dirty. It allows you to dive in and start using chef quickly rather than worrying about setting up a chef server instance which would be yet another instance to feed and water.

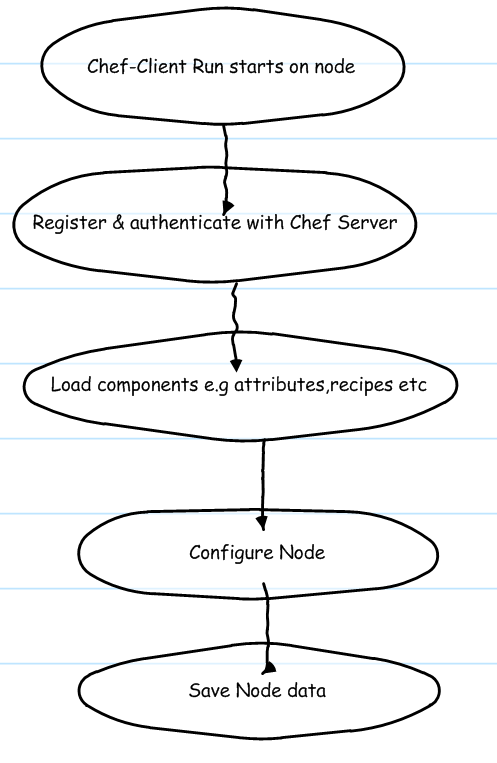
Next you will have to set up an admin workstation(s) environment from where you will create & edit cookbooks and use the command line tool knife. We will be using a windows 7 workstation. To set up your windows workstation be this on premise or an instance on AWS follow the instructions outlined here : <http://devopscloud.net/2011/04/17/managing-chef-from-windows-7/>

So assuming you have done this and registered with Opscode or set up your own Chef server the next thing to do is determine what you want chef to actually do to the instances you fire up via CloudFormation.

We would suggest creating a flow chart and actually undertaking the various steps outside of chef on a target instance. Doing this will l help when debugging recipes.

To become productive with chef with windows target nodes quickly use powershell as much as possible. This means the effort when using Chef to mange windows nodes is really down to understanding enough chef and ruby to carry out basic flow and to call out to do simple actions like running powershell scripts , installing MSI’s and a bit of process flow management. As a windows administrator it is assumed a certain basic foundation in Powershell.

The diagram below outlines what happens on a chef-client run at a high level . A more detailed description of exactly what happens can be found at <http://wiki.opscode.com/display/chef/Anatomy+of+a+Chef+Run>



003-82

Both windows and Linux need a little pep work for some pre-reqs that will allow them to install the latest stable chef gem and then do the chef-client run.

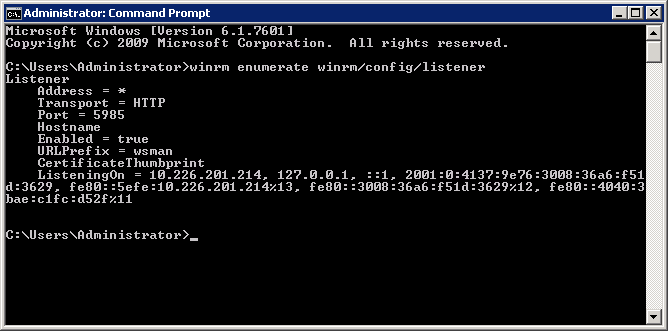
There is a bootstrap command available called knife that does not require as much prep work but this will only be of use for post initial provisioning & configuration. We are outlying an end to and automated provisioning and configuration solution so knife is not appropriate as we will be using CloudFormation and not a workstation that has chef and knife configured on it to kick off the creation of the environments.

For a windows instance the prep work for the AMI this needs the following steps as a minimum. Windows is slower to be in a ready state on ec2 than Linux instances so anything that can be done to give it a helping hand should be baked into the AMI.

1. Change the default password to something secure and memorable. This is described in the section ‘Configuring the development environment.
2. Make sure WinRM is set up. This is on by default on windows 2008 R2 and can be checked by enumerating using the following command:

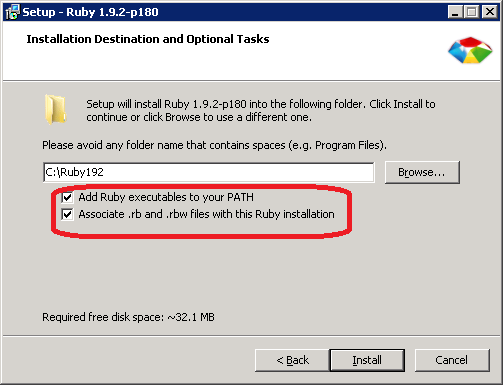
winrm enumerate winrm/config/listener

The diagram below shows the successful confirmation that WinRM is configured:



003-84

1. Set Powershell execution policy to Remote Signed
2. Install the latest version of ruby ( at the time of writing this was 1.9.2-p180) from <http://rubyinstaller.org>. Note it is worth double checking that the version of ruby is compatible with the latest stable release of Chef. Make sure that the ruby executables are in the path and .rb files are associated with the ruby installation as illustrated in the diagram below:



003-85

1. Create the following folders: c:\etc c:\etc\chef and c:\devkit
2. Install the Ruby Devkit

Download from <https://github.com/oneclick/rubyinstaller/wiki/Development-Kit>  the ruby development kit

Extract the  devkit  into c:\devkit by copying the downloaded devkit.exe into c:\devkit, then  extracting it using  DevKit-tdm-32-4.5.1-20101214-1400-sfx.exe –y

Then run the following

ruby c:/DevKit/dk.rb init

ruby c:/DevKit/dk.rb install

1. Install some pre-req gems using the command

gem install  ruby-wmi windows-api windows-pr

1. If you have set up your Chef environment then you will then need to copy over client.rb and also yourorganisation-validator.pem into the c:\etc\chef folder these will allow the instance to register with the chef server hosted at Opscode.
2. Finally a little prep work is needed to allow the windows instance to automatically install chef and then do a chef-client run on start up. We describe one of many options that uses PowerShell .

Create a PowerShell script that has the following contents:

|  |
| --- |
| #setupchefClient.ps1  # install chef gem - This ensures only the latest stable version is installed  $installchef= "gem install chef --no-rdoc --no-ri""  # grab userdata  $webclient = new-object system.net.webclient  $awsurl ="http://169.254.169.254/latest/user-data "  $targetfile ="c:\etc\chef\runlist.json"  $webClient.DownloadFile("$awsurl","$targetfile")  # Run chef-client passing json file with runlist  $runchef = "C:\Ruby192\bin\chef-client -j"+ $targetfile  # Write-Host "Waiting 10 seconds"  Start-Sleep -s 10  invoke-expression $installchef  invoke-expression $runchef |

Copy the script into the c:\etc\chef folder or another location

This script uses the userdata that it will be passed as part of it’s initial start up . The script can be wrapped as a service or maybe use the runonce registry setting to be run on initial startup

The userdata passed to the instance contains the chef run list that is set up on your opscode server and will look similar to the below

|  |
| --- |
| {"run\_list":["role[webserver]"]} |

Once you have an instance configured with the pre-reqs you then need to save this as an AMI

In the above run list example we are applying a role to the target server .This role groups together cookbook recipes that define the actions that are needed to configure the instance for it’s eventual role as a windows webserver.

To configure a set of windows web servers using chef the cookbook needs to be configured initially. The post here <http://devopscloud.net/2011/03/23/dissection-of-a-chef-recipe-or-two-for-windows/> explains how to use chef to create simple recipes against a windows target.

Setting up a service or daemon or running the chef client with the –i flag post the initial start up run will make sure that any updates to the cookbooks are applied without having to actually log onto the target nodes or use the knife command.

Using CloudFormation and Chef to configure ec2 instances

With Cloudformation to provision the instances and Chef to configure them the next step is to use them together to provide an end to end provisioning and configuration of the target ec2 instances.

Opscode has a command line tool called knife that allows you to create and manage your target nodes by pushing the changes to the target. As we will be kicking of the chef run as part of the cloudformation provisioning steps we will need the node to do the asking once it has been provisoned. This is achieved by getting the node to do a chef-client run on start up.

In the section on using chef to configure AWS resources it was described how to capture user-data passed to a windows instance and create a json file that contains the run list to be applied to the target node as part of a chef-client run.

The key to linking CloudFormation and chef together to achieve an automated provisioning and configuration is to be able to pass the user-data to an AMI that has been suitably prepared as described earlier to use the user-data.

To do this you need to do the following in your CloudFormation script that is used to start instances

Create a parameter for the run list:

|  |
| --- |
| "RunList": {  "Default": "role[basewindows]",  "Description" : "Chef Role ",  "Type": "String"  } |

Add a UserData property that looks like the one shown to the ec2 instance resource

|  |
| --- |
| "UserData": {  "Fn::Base64": {  "Fn::Join": [  "",  [  "{\"run\_list\":\"",  {"Ref": "RunList"},  "\"}"  ]  ]  }  } |

On the windows Instance it needs to be able to use this data to create a runlist JSON file and run it.

A Powershell script similar to the one below achieves this. This script can be wrapped up as a service or set to run once via the RunOnce Registry setting so that on start up the script is run and the instance is configured.

|  |
| --- |
| #chef-clientrun.ps1  $installchef= “gem install chef –no-rdoc –no-ri”"  # Download userdata  $webclient = new-object system.net.webclient  $awsurl =”http://169.254.169.254/latest/user-data ”  $targetfile =”c:\chef\etc\runlist.json”  $webClient.DownloadFile(“$awsurl”,”$targetfile”)  # Run chef-client passing json file which contains runlist  $runchef = “C:\Ruby192\bin\chef-client –j ”+  $targetfile  invoke-expression $installchef  invoke-expression $runchef |

Monitoring using CloudWatch

Cloudwatch metrics are enabled by default sampling metrics at 5 minute intervals for ec2. CloudWatch also monitors Elastic Load Balancers , EBS volumes and RDS DB instances .

Detailed metrics can be enabled at a cost at the time of writing of  $0.015 per instance-hour , this allows sampling at 1 minute intervals and enables data aggregation by AMI ID and instance type.

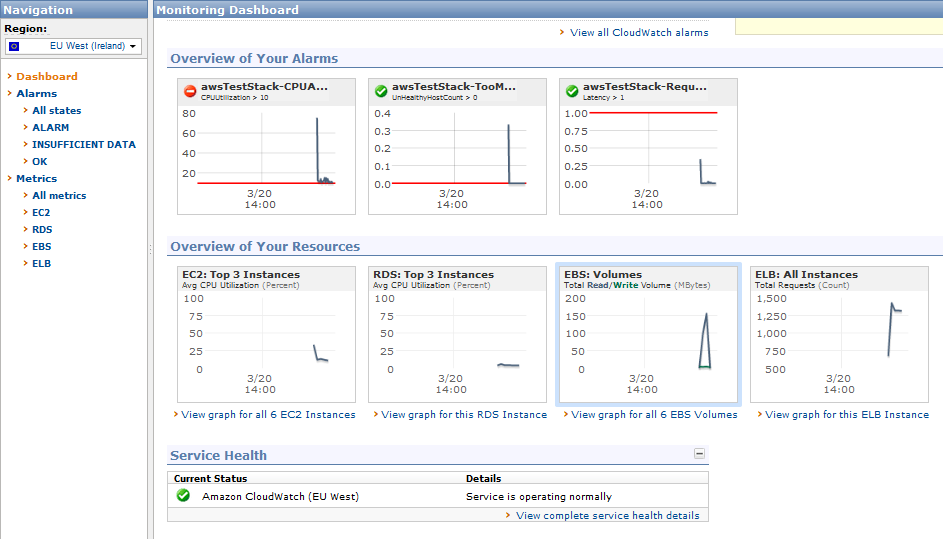
If Auto Scaling or Elastic Load Balancing is being use CloudWatch will also provide EC2 metrics aggregated by Auto Scaling group and by Elastic Load Balancer, regardless of whether Basic or Detailed Monitoring has been selected.

Monitoring data is persisted for 2 weeks, even if your AWS resources have been terminated

**Viewing metrics and Setting up alarms using the console**

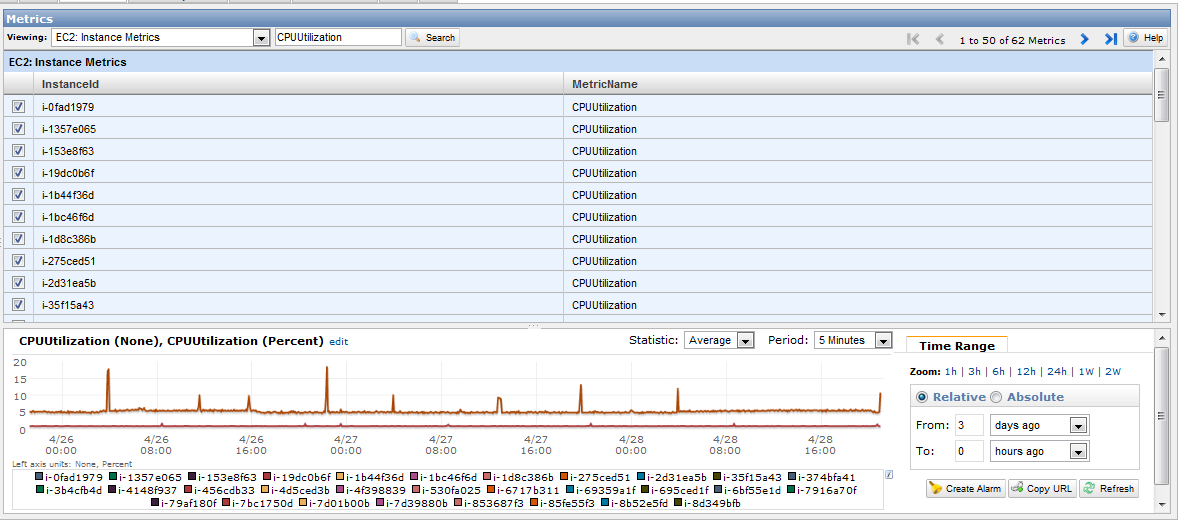
From the console selecting the Cloudwatch tab provides access to a dashboard that provides views of alarms and metrics for your AWS estate.

The first initial view is the over view as illustrated in the screen shot below:



004-01

From here you can drill down to obtain a more detailed analysis and It is also possible to look at aggregated data across a anumber of instances



004-06

Using alarms allows AWS to automatically take actions that will be effected based on threshold values that are defined for a specific metric over a number of time periods .

Alarms can be set from the console , via the command line and of course via an AWS stack creation.

The walkthrough below illustrates how to set up an alarm to undertake an action when RDS storage reaches a specified threshold level using the console. This assumes that an RDS instance has already been created

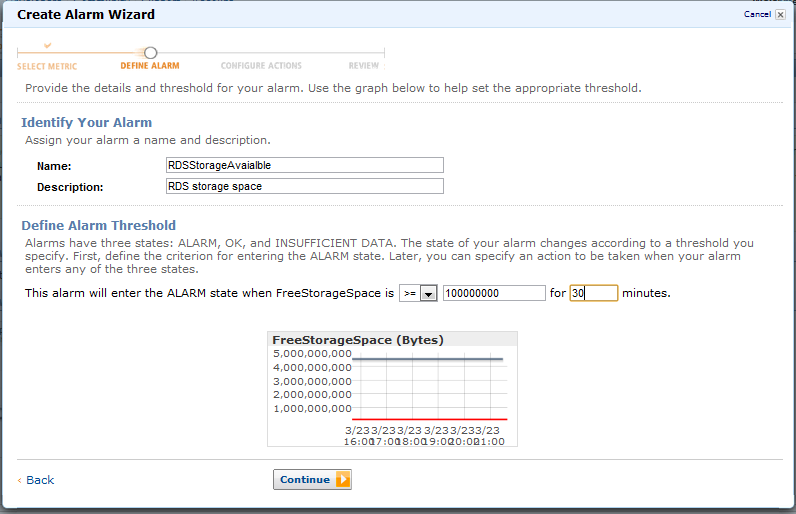
Select the CloudWatch tab then click on­­ the Create Alarm button



004-07

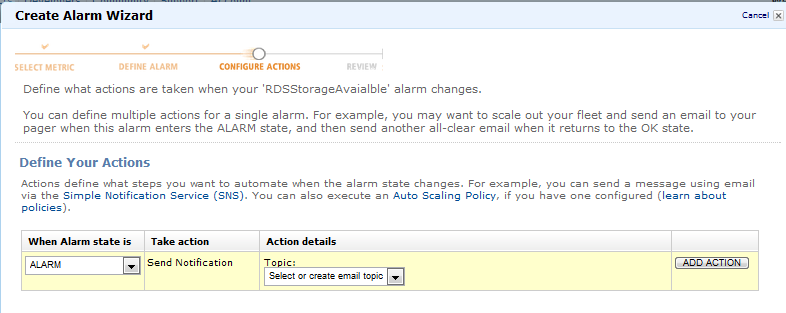
Select RDS: Database metrics from the drop down list.

Then select the FreeStorageSpace metric and click continue. The next form is where the parameters needed to define the alarm state are set. So in the case of the metric selected we are setting the value for when free space drops to a specific level



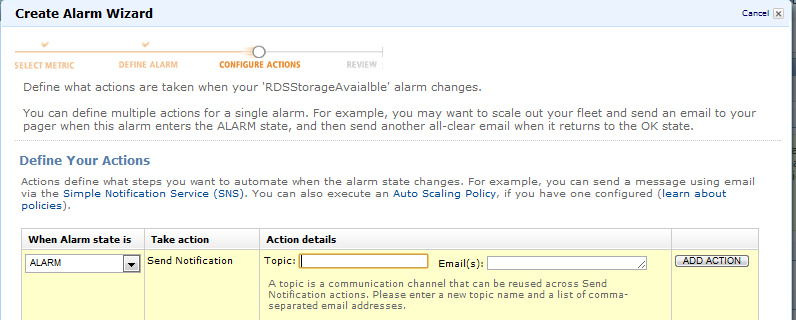
004-08

Click continue to get to the form where actions are configured for the alarm.



004-09

An SNS topic can then be created



004-10

**Setting Alarms using CloudFormation:**

To use CloudFormation to set up alarms requires setting up a resource of the type “AWS::CloudWatch::Alarm”. The snippet below shows an alarm associated with monitoring the CPU usage

|  |
| --- |
| "CPUAlarmHigh" : {  "Type" : "AWS::CloudWatch::Alarm",  "Properties" : {  "AlarmDescription" : "Alarm if CPU too high or metric disappears indicating instance unavailable",  "AlarmActions" : [ { "Ref" : "AlarmTopic" } ],  "InsufficientDataActions" : [ { "Ref" : "AlarmTopic" } ],  "MetricName" : "CPUUtilization",  "Namespace" : "AWS/EC2",  "Statistic" : "Average",  "Period" : "60",  "EvaluationPeriods" : "1",  "Threshold" : "10",  "ComparisonOperator" : "GreaterThanThreshold",  "Dimensions" : [ {  "Name" : "AutoScalingGroupName",  "Value" : { "Ref" : "ApplicationServerGroup" }  } ]  }  }, |

The AlarmTopic Reference referred to is an SNS topic

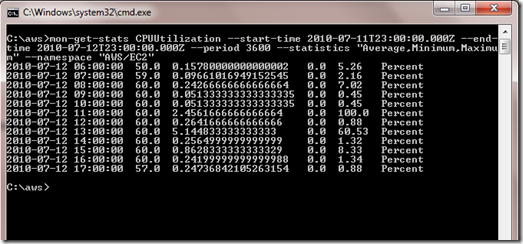
|  |
| --- |
| "AlarmTopic" : {  "Type" : "AWS::SNS::Topic",  "Properties" : {  "Subscription" : [ {  "Endpoint" : { "Ref" : "AdminEmail" },  "Protocol" : "email"  } ]  }  }, |

**Viewing metrics using the Command line tools:**

With the API tools you can use the command line tools to see the list of metrics exposed via this.  Once set up you have the ability to run fairly ugly commands such as the example below where we are collecting the CPU stats from an EC2 estate ( In this case a single instance)

mon-get-stats CPUUtilization --start-time 2010-07-11T23:00:00.000Z --end-time 2010-07-12T23:00:00.000Z --period 3600 --statistics "Average,Minimum,Maximum" --namespace "AWS/EC2"

Which spits out the metrics:



Not pretty but quick and dirty and gets the job done. You can pretty much understand why wrapping this stuff up is a must do really.

**Using 3rd party monitoring tools:**

As an architect or developer it is unlikely that you will be responsible for the ongoing maintenance of the application and it would be prudent to engage with your operational team at the earliest opportunity to find out what tools they already use to monitor and to see if you can surface the cloudwatch metrics within their tool and expose application level metrics.

If you are fortunate to be in a position to recommend something then find out how the operational team works today and suggest something that will fit in with their existing operational procedures. It cannot be stressed enough how important it is to engage with the operational team as early in the development cycle as possible.

There are tools such as PRTG <http://www.paessler.com/amazon_cloudwatch> which are able to consume Cloudwatch metrics but there are also tools such as NewRelic <http://newrelic.com/> that complement the CloudWatch metrics by being able to provide greater depth to the monitoring by providing application level monitoring .

It would be remiss not to mention AWS monitoring solutions for mobiles. Most people have their phone close by most of the time so using a phone as an integral part of a monitoring solution makes sense One such application is Decaf <http://www.9apps.net/decaf-ec2-client/> which is available for the android mobile O/S.