Cloudformation servisi temel bazı AWS servisleri öğrenildikten sonra anlatılıyor. Bazı kurslarda kurs sonunda verilir. Ancak projelerde kullanabilmemiz için bu servis önden veriliyor.

Markette bu ve buna benzer toolları kullanmamız bekleniyor. Cloud ve Devopsçu olarak templateleri yazmamız gerekir.

Resource oluştururken konsoldan UI de oluşturuyorduk.

Cloudformation la bunu yapabiliriz. Ancak cloudformation bunu yapabilmek için template e ihtiyaç duyar. Biz buna cloudformation template diyoruz. Daha önce Launch Template miz vardı. AMI de bir çeşit template ti. Bu da son template miz olacaktı.

Launch template sadece bir EC2 ayağa kaldırıyordu. AWS bizim için bu templatei kendi sisteminde saklıyor, biz bunu indiremiyorduk. AMI de EC2 nun software kısmının template i demiştik.

Şimdi de cloudformation template i görecez. Burada yalnızca bir EC2 değil diğer servisleri de içinde barındırabilen bütün bir mimariyi ayağa kaldırabiliyoruz.

Buna sektörde infrastructure as code denir. Cloudformation bunlardan birisidir. Bir de terraform göreceğiz. Cloudformation sadece AWS servisleri için kullanılırken terraform Azure gibi diğer cloud providerlerın servisleri için de kullanılır.

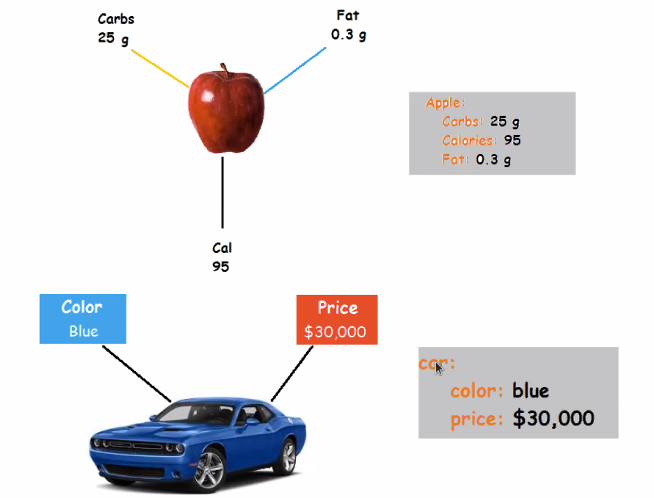
Cloudformation templatelerin .yaml veya .json formatında olması gerekir.

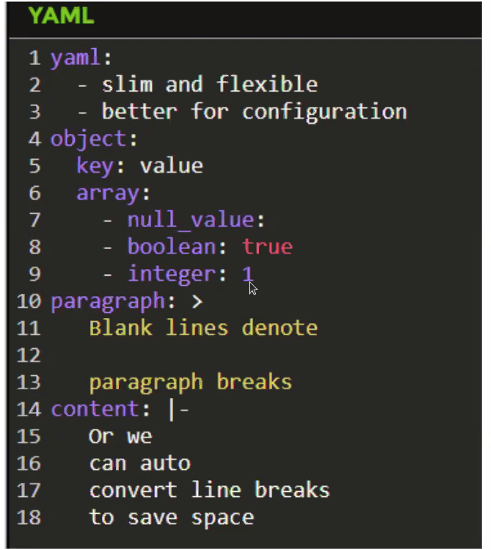
Bugün bir intro session olacak. Cloudformation a ileride detaylı girecez.

YAML ve JSON veri tutma formatlarıdır.

YAML dan başlayalım;

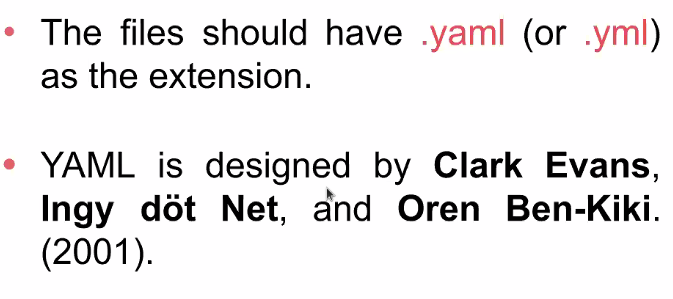
Bir obje tanımlayıp bununla ilgili özelikleri bu programlar (yaml ve json) aracılığıyla tutabiliyoruz.





Data-serialization; hazırladığımız yazılımda ya da oluşturduğumuz mimaride hazırladığımız objelere ilişkin verileri başka bir yerde de kullanma ihtiyacımız oluyor. Bu programlar ya da serverlar arasındaki bu veri paylaşmının genel ismi data-serialization dur.

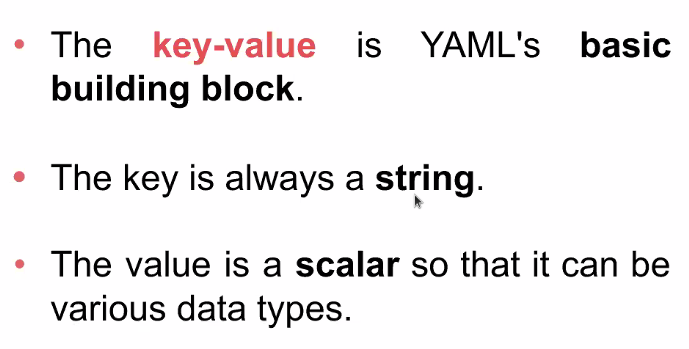
Yaml genel anlamda configuration için kullanılır. Ancak data-serialization için de kullanılır. Daha çok tercih edilir. Çünkü daha human-readable dır.



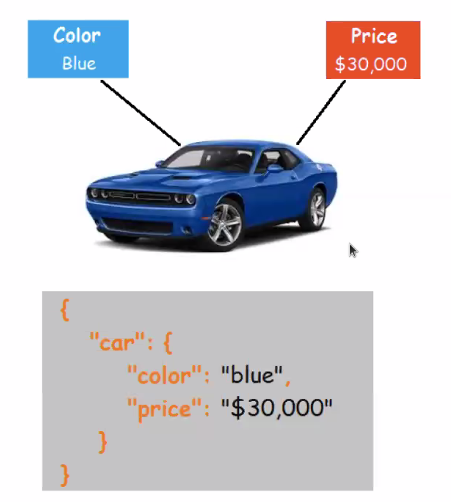
<https://yaml.org/>

Daha önceleri .xml formatı kullanılıyordu. Şu anda jason ve yaml piyasayı domine etmiş durumda.

Yaml da veriler key-value (anahtar-değer) şeklinde tutuluyor. Key değerleri string olmalı valu değerleri scalar dır.



JSON ise daha çok veri transferi (data-serializaiton) için kullanılıyor.



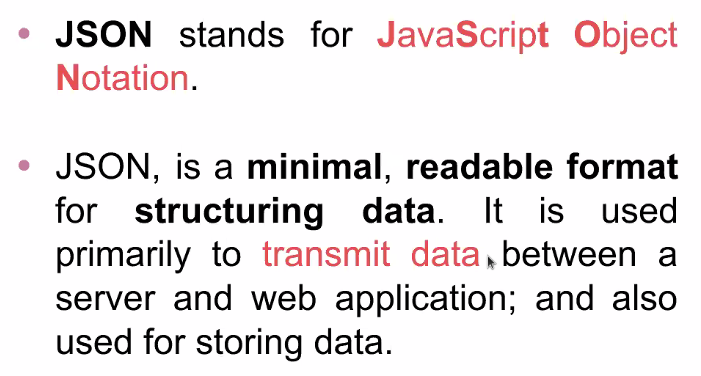


Örneğin EC2 ya S3 te bucket replication u yaptırırken JSON ile yapmıştık.

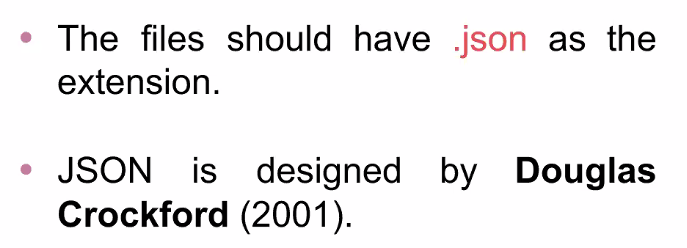
Bunlara programların ve serverların arasındaki konuşma dili diyebiliriz. İlave olarak; bir program C dilinde diğeri Python dilinde yazılmış olsun. Bunlar birbiriyle haberleşirken JSON veya YAML kullanır. (XML de)

JSON da görülen şekilde objeler {} arasında oluşturuluyor. Arraylar [] ile oluşturuluyor. JSON da stringler "" içinde olmak zorunda.

YAML da objeler, keyler arasınaki ayrım girintiyle olurken (indentation) JSON da , ile {} ile olur.

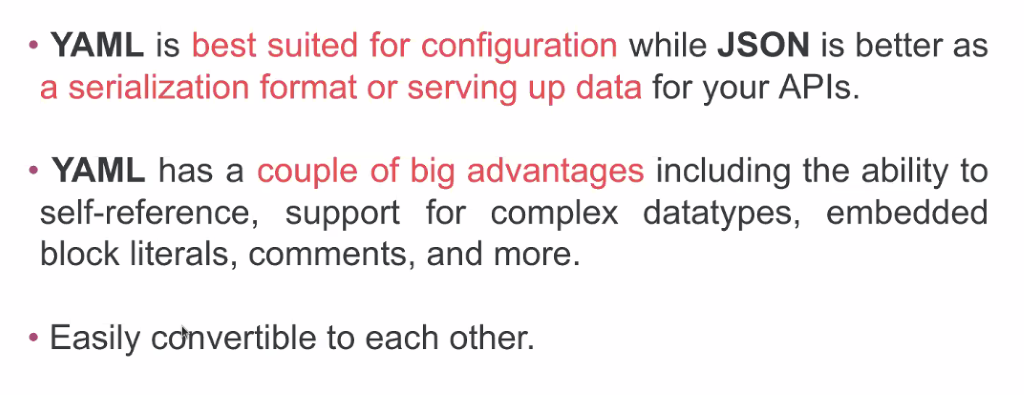


Değişik veri tiplerindeki veriyi bir bütün halinde tutabilir JSON. dosyalar .json formatındadır.



<https://www.json.org/json-en.html>

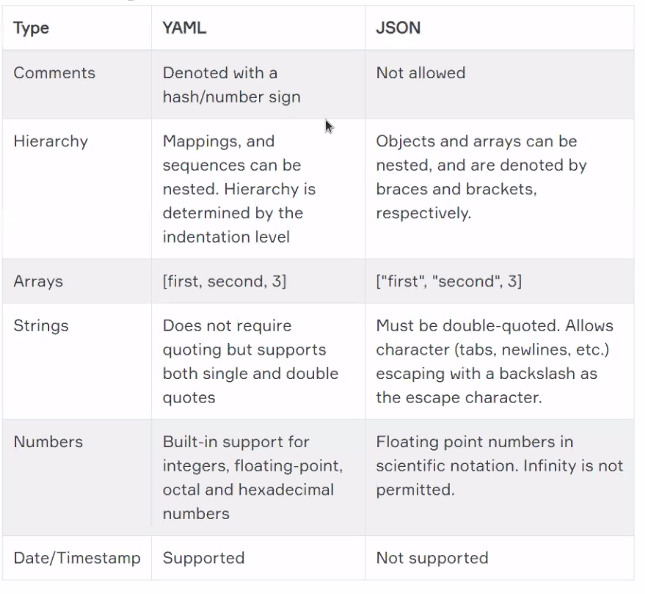
YAML ve JSON arasında kısa bir karşılaştırma:



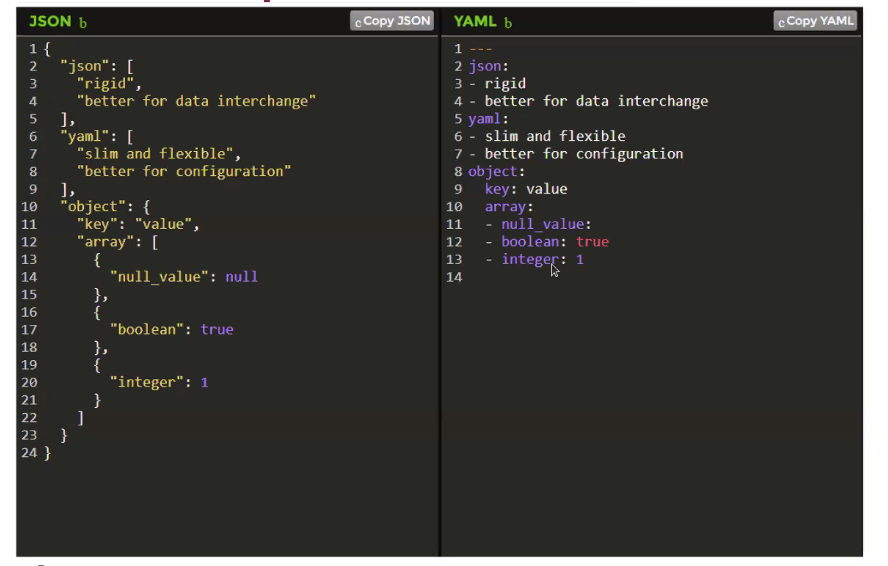
YAML daha çok configuration için (Kubernetes, vs.) JSON daha çok yapılar, serverlar, app ler arasında veri transferi için kullanılır.

YAML formatı içinde referans sistemi kullanabiliyoruz. Complex data typlerı destekler (date, inf gibi) Commentler yazabiliriz. JSON da bunlar mümkün değil.

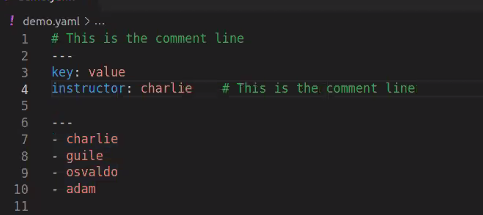
Her ikisi de birbirine kolaylıkla dönüştürülebilir. (online programlar ve Vs code extensionlarla)



YAML da hiyerarşi indentationla olurken JSON da objeler ve arrays ler nested olabilir ve { [ ile gösterilirler.

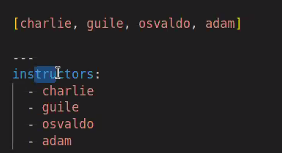


aynı içeriklerin JSON ve YAML da görünümleri.



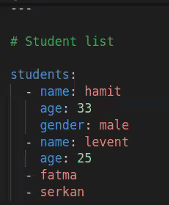
üç tire - - - farklı yapıları ayırmak için kullanılır. - - - diye ayırmazsak yapıları ayrı dosyalarda tutup ayrı configure etmemiz gerekir.

listeleri şu şekillerde gösterebiliriz:

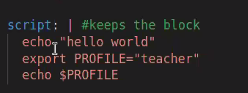


Key in valuesi nested halde key ler, listeler de olabilir:





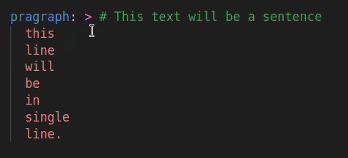
şu işaret; | arkasından gelen verileri satır satır ele almasını (parse etmesi - okumasını) sağlar:



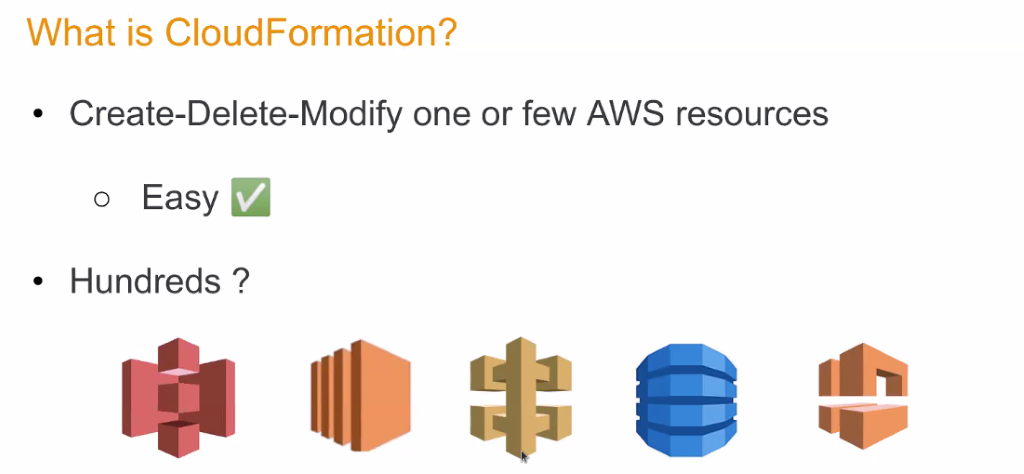
YAML da tab da sıkıntılar çıkıyor farklı serverlar arasında tab ayarı farklı olduğu için. kullanılmazsa iyi olur.

Biz genel olarak | user datalar da kullanıyoruz.

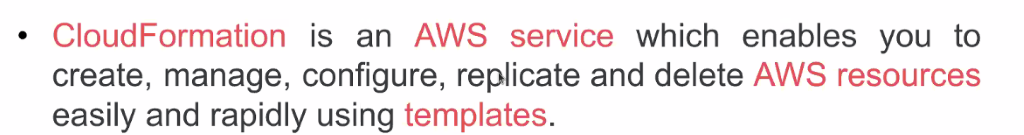
Bir de > işareti var. Ayrı ayrı satırları tek bir satır halinde gör demektir:



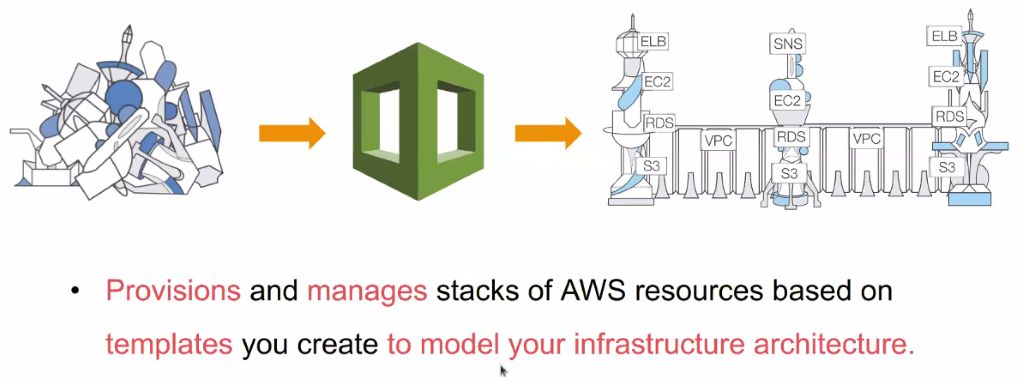
CLOUDFORMATION:



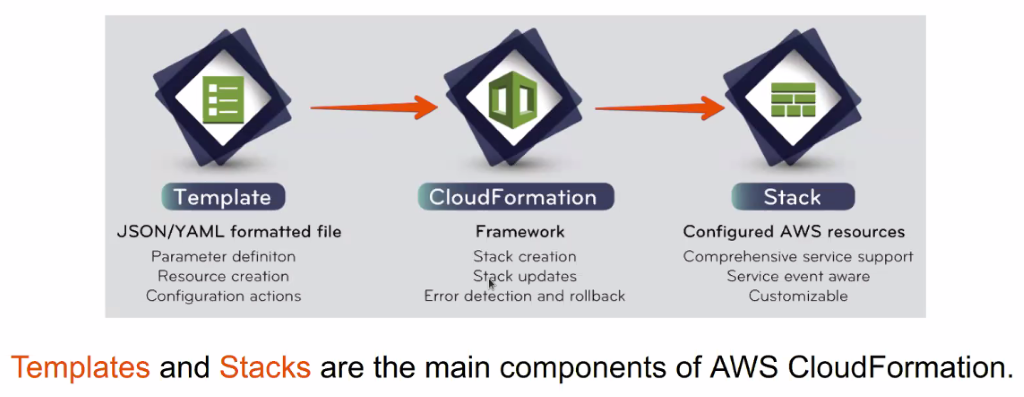
Tek bir instace ya da server ayağa kaldıracakken konsol kullanılabilir. Ancak bütün bir mimariyi, EC2 lar, databaseler, S3 ler IAM ler configure derken konsol kullanmak çok uğraştırıcı olur. Cloudformation bu işi apar. Template dosyalarını bu servise verdiğimizde template içerisinde resource lara göre komple mimariyi ayağa kaldırır.



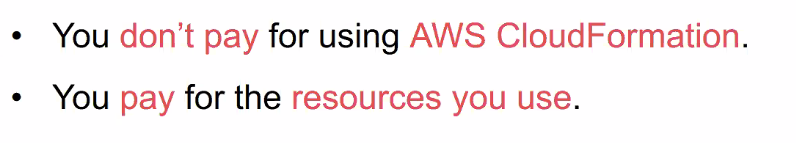
Hazırdaki bu templateler birden fazla kez kullanılabilir.



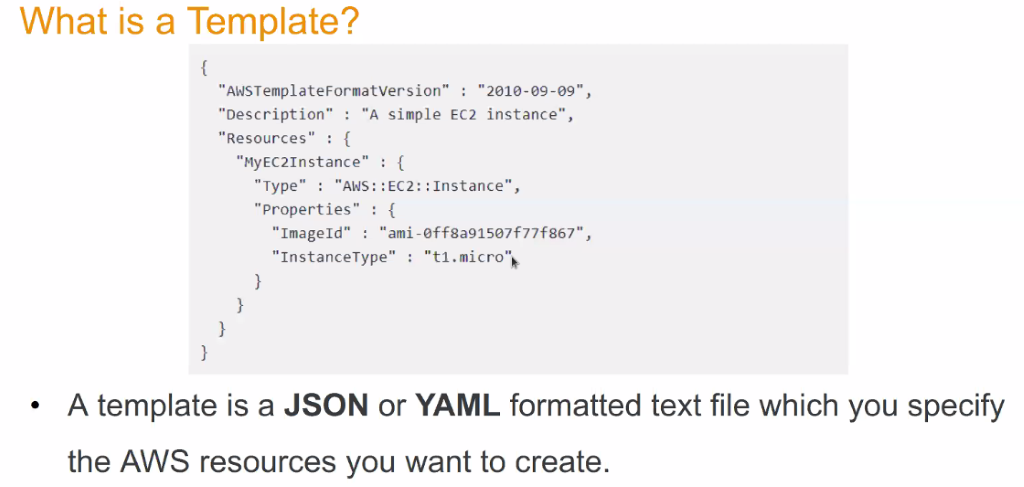
Bu mimariye stack denir.



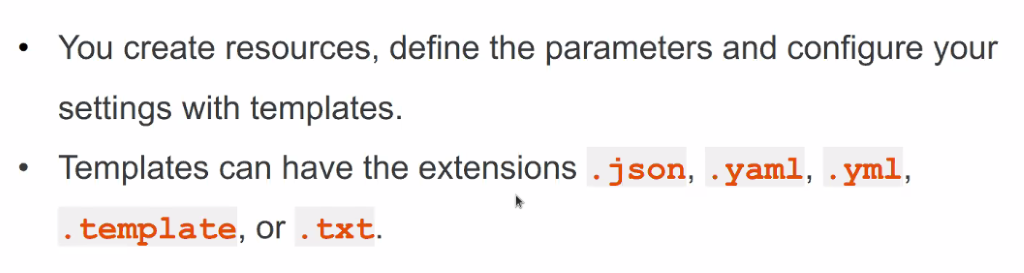
Template ler ve Stackler AWS CloudFormation un temel bileşenleridir.



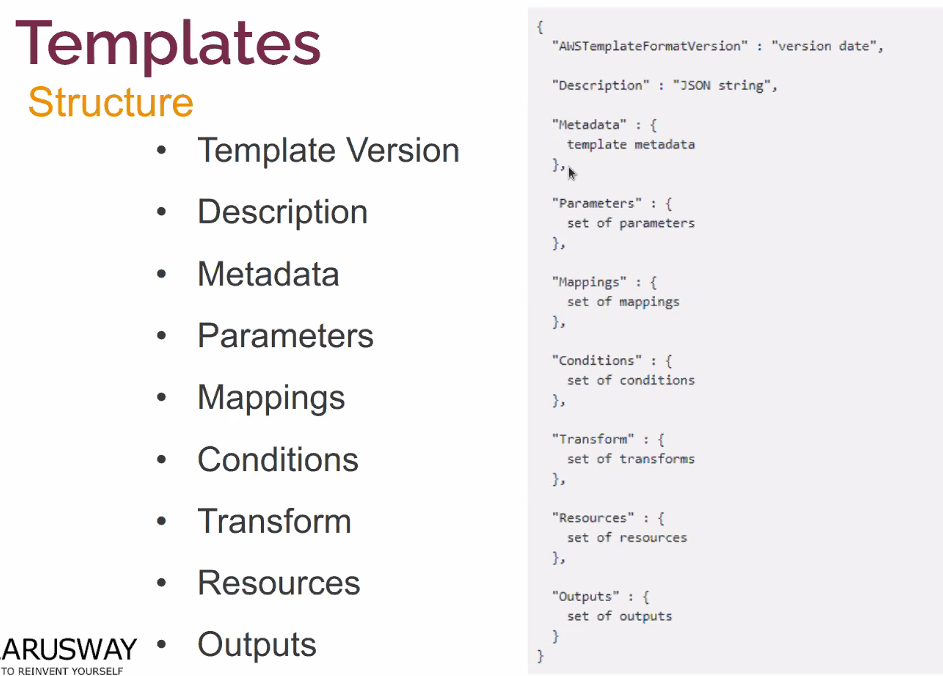
CloudFormation kullanmak ücretli değildir, içindeki servislerin ücretleri varsa onlar ödenir.



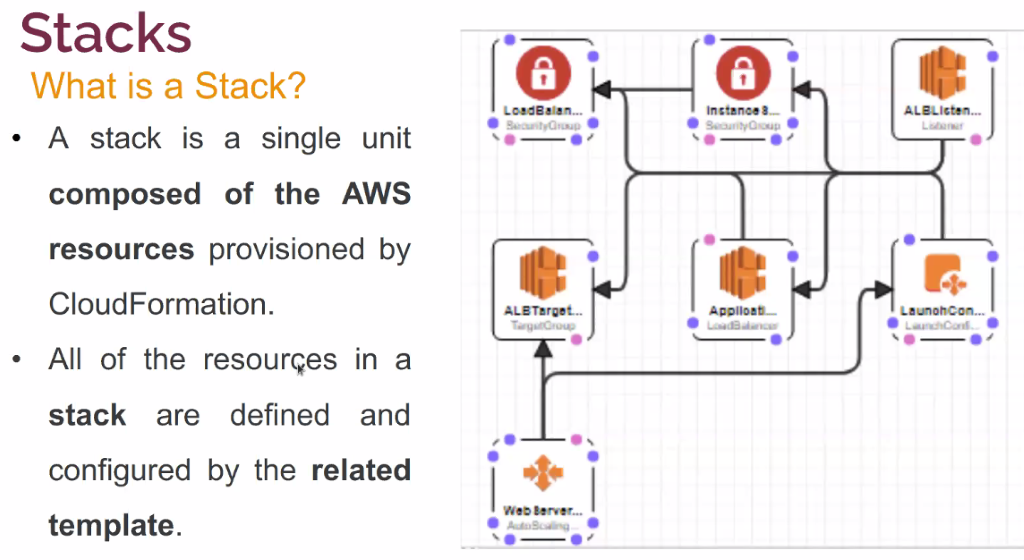
Templateler JSON ya da YAML formatında hazırlanıyor. Template in nasıl hazırlanacağı ile ilgili, key ve value lerin neler olacağını AWS dokümantasyonunu kullanmamız gerekiyor.



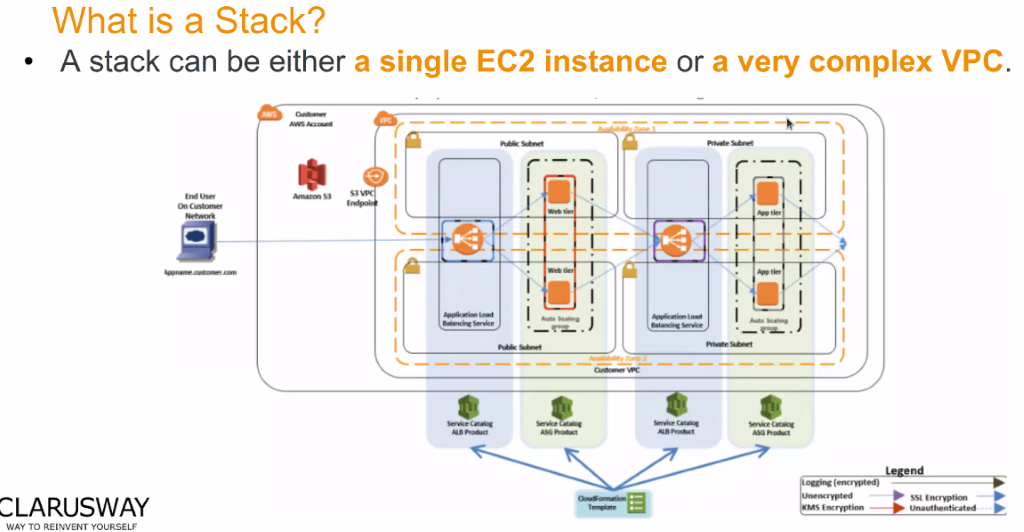
template uzantıları .json, .yaml, .yml, ,template, .txt olabilir. Bir metin dosyası içinde de template yazılabilir. Ancak içerisindeki veri formatı JSON yada YAML formatında olmalı.



AWS template içeriğini bu şekilde zorunlu tutar. Burada dolu olması zorunlu olan yer ise Resources alanıdır. Bu alanda gerekli verileri girersek CloudFormation bu mimariyi oluşturur. Resources u olmayan bir template çalışmaz. ancak diğer kısımlar olmazsa çalışır.

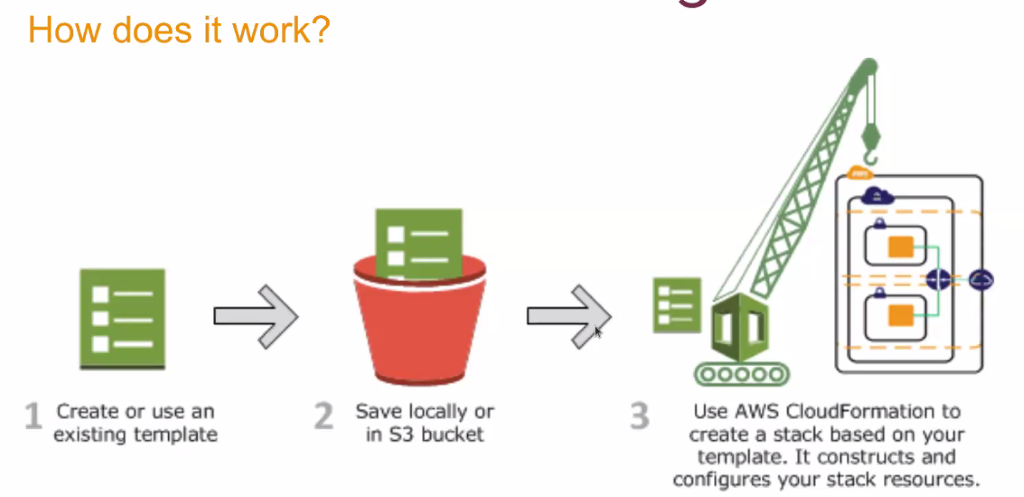


Stack template te tanımadığımız resources lar bütününü tamamını ifade eder. Mimariyi ayağa kaldırırken bir bütün olarak ayağa kaldırdığı gibi silerken de bir bütün olarak siler.



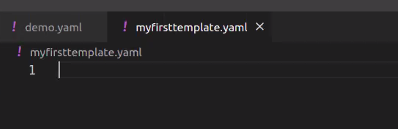
Bir stack tek bir EC2 olabileceği gibi çok complex bir VPC de olabilir.

Template te hata varsa CloudFormation bizi uyarıyor. Biz de o ilgili kısımdaki hatayı düzeltiyoruz.



Hazırladığmız template i lokalden yükleyebileceğimiz gibi S3 te te tutabiliriz. Ancak template kullandıktan sonra AWS bunu S3 te bizim adımıza saklar.

Bir template dosyası oluşturuyoruz:

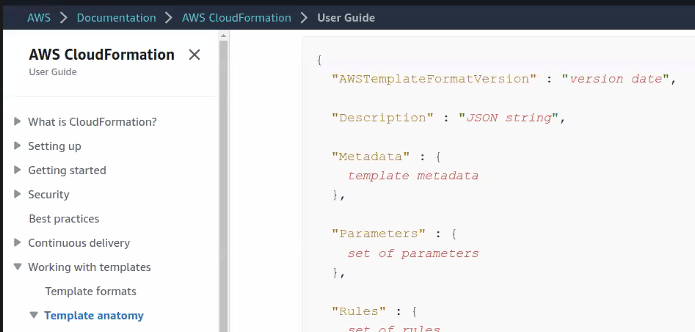


YAML dosyası olduğu için .yaml uzantılı olacak.

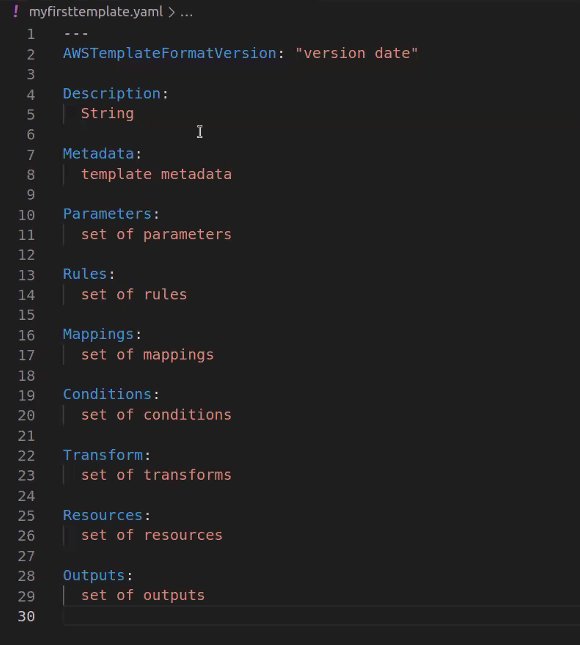
VS Code daki extenisonlar bize yardımcı oluyor.

AWS dokümantasyon saayfasında Template anatomy sayfasında format mevcut:

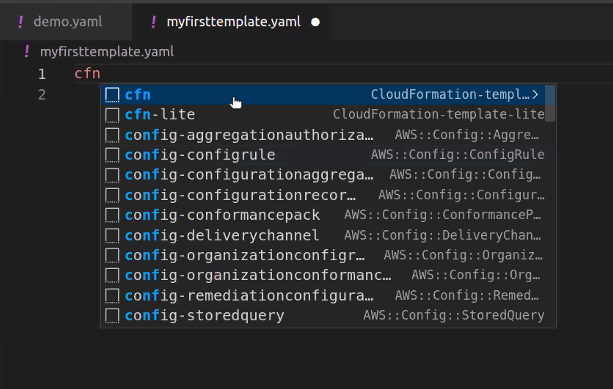
<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/template-anatomy.html>



Oradan alıp dosyamıza kopyalıyoruz:

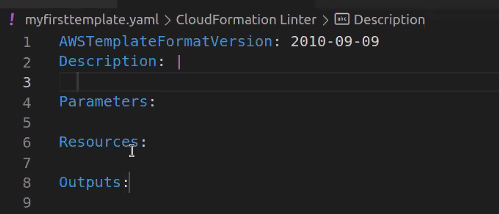


cfn yazdığımızda cloudformation extension u gelir:



buradan da template anatomisini oluşturur.

cfn-lite de ana başlıkları getirir:

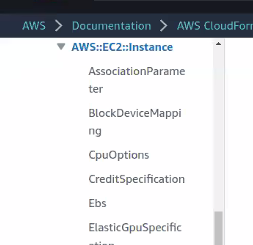


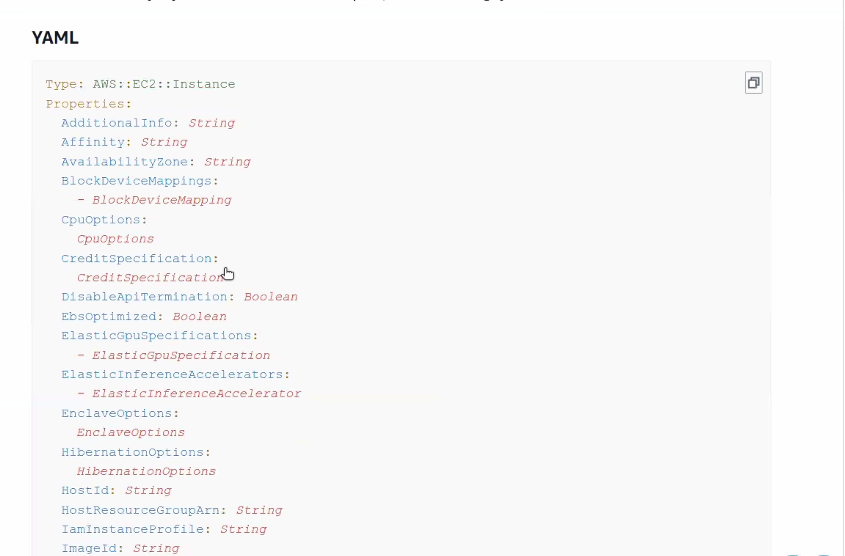
Description da açıklama yazarız.

Kullanıcılardan herhangi bir girdi girmesini bekliyorsak bunu parametreler kısmında yazmamız gerekir. Bunu sonra göreceğiz.

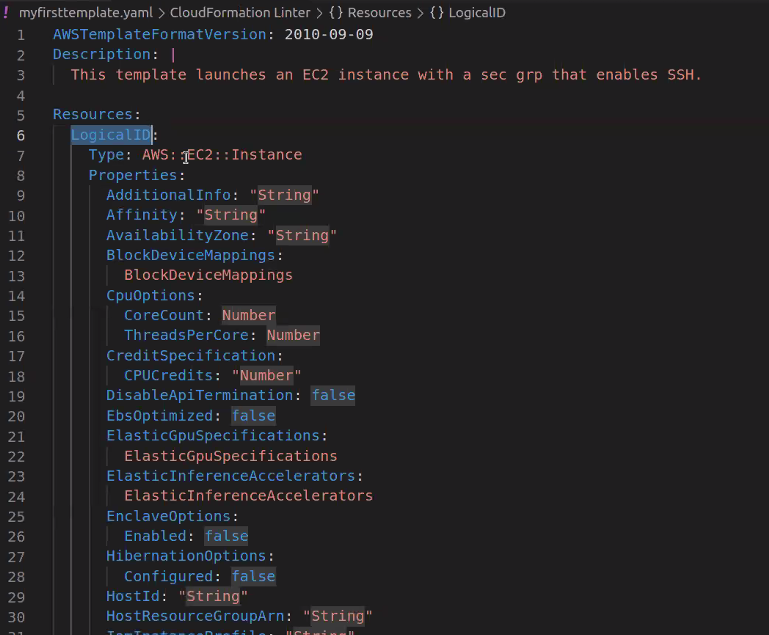
Outputs; EC2 ayağa kalktıktan sonra görebildiğimiz public DNS ve IP gibi bilgileri Cloudofrmaiton da görebilmek için buraya gerekli verileri griyoruz. Sonra göreceğiz.

Resources bölümünde ayağa kaldıracağımız servisleri yazacağız. Hangi servisi yazmamız gerkiyorsa onunla ilgili de dokümantasyonda bilgi mevcut.



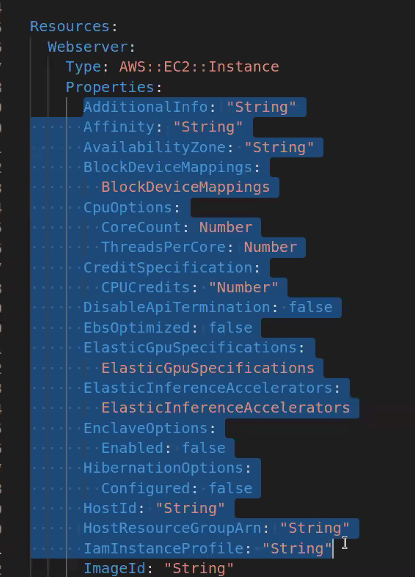


Ancak buradn kopyalamıyoruz. Vs code daki uzantılar sayesinde EC2 instance yazdığımızda bize öneri olarak getirecektir:



Logical ID: kendi belirlediğimiz bir isim olabilir bu. Web Server diyoruz. (Database server vs.) Başka bir resource bu template i kullanmak isterse bu logical id üzerinden tanımlama yapar. Bu instance ın variable değeri gibidir.

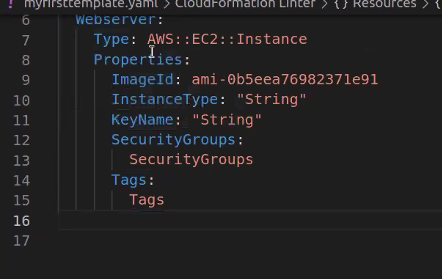
Bu taslak bir EC2 ayağa kaldırmak için gereken tüm bilgileri içeriyor. Girmediğimiz verileri default olarak belirleyecektir.



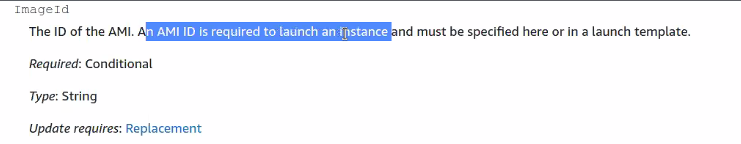
Bu bilgileri istemediğimiz için Image id ye kadar siliyoruz. Bu Image id konsoldan image in id sini alıp yazmamız gerkiyor.

Belirlemeyeceğimiz diğer verileri de silmemiz gerekir. Aksi takdirde "string" şeklinde kalırsa hata alırız.

Şimdilik şu key leri tutuyoruz:

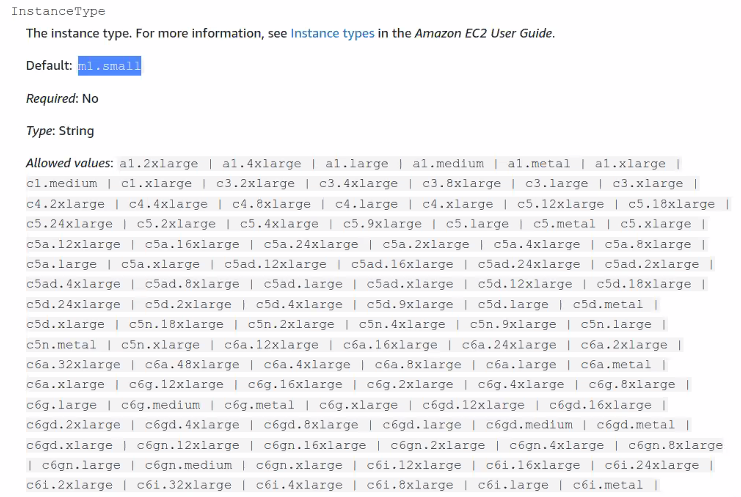


Örneğin dokümantasyona gidip ImageId nin detayına baktığımızda nasıl yazmamız gerektiğiniz bize böyle anlatır:



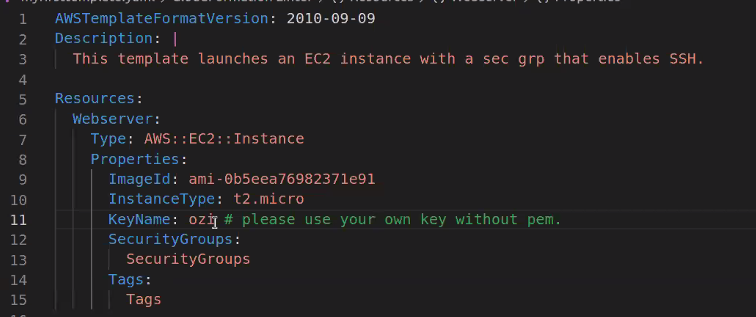
CloudFormation da regional bir servistir. Hangi regionda kullanacaksak o regiondaki AMI id yi almamız gerekir.

Dokümantasyonda Instancetype ın bilgilerine bakalım:

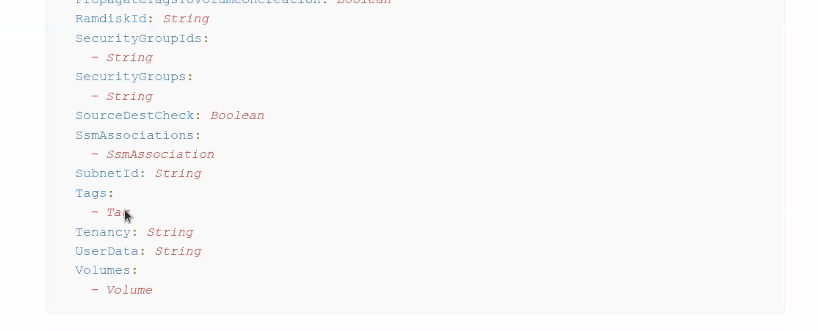


default olarak m1.small ı gösteriyor. Instancetype ı da sileresek template ten default olarak bu type ı alacak. Altta da olabilecek seçenekleri gösteriyor. Gösterildiği şekilde yazmamız gerekiyor. Biz t2.micro yazıyoruz.

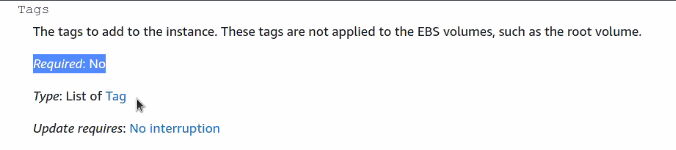
Keyname mevcuttaki keyname i pem olamdan yazarak kullanacağımız gibi burada ayrıca bir key de oluşturabiliriz. Bu template i başkaı kullanırken kendi keyname ini yazması gerektiği için buraya comment yazıyoruz:



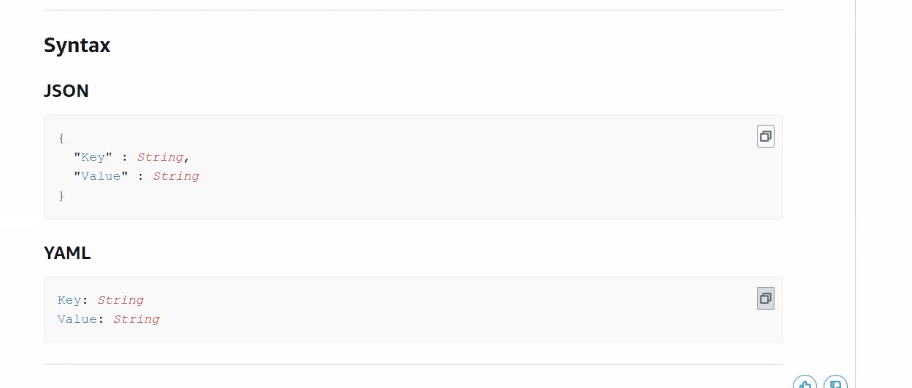
Tag i nasıl vereceğimizi dokümnatasyondan inceleyelim:



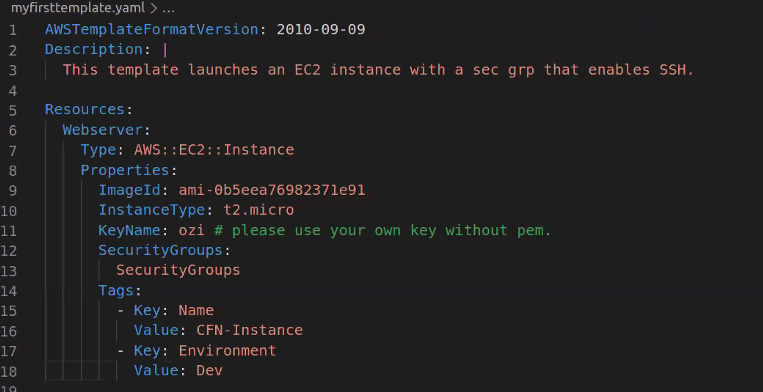
şu gösterim: - Tag list şeklinde yazılması gerektiğini söyler. detayına bakalım:



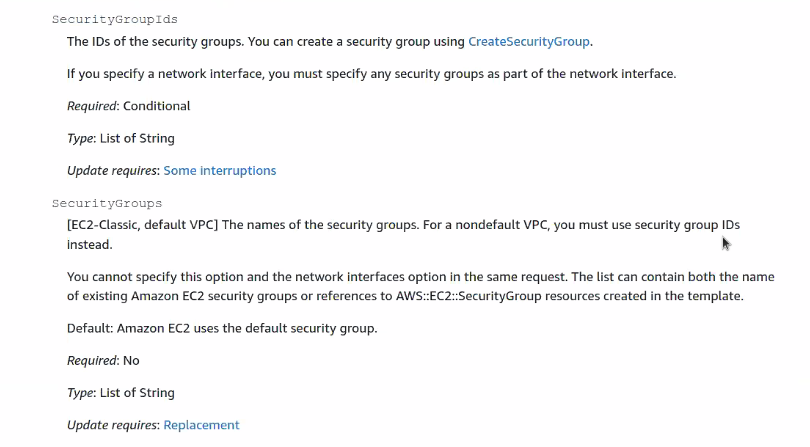
Tag e basınca da nasıl yazmam gerektiğine dair syntax i bize gösterecek:



Şu şekilde iki Tag ekleyelim:



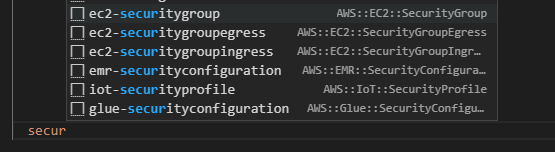
SceurityGroup larımızdan bir security group ismini buraya yazarak kullanabiliriz. Ancak biz yeni oluşturacaz aşağıda. Dokumantasyona bakalım.



Default VPC ler için securtiy groups kullanabiliriz ancak default olmayan VPC ler için SecurityGroupID key ini kullanmamız gerekiyor.

SecurityGroup u list olarak yazmamız gerekiyor. Hiç vermezsek AWS default olarak security group verecek.

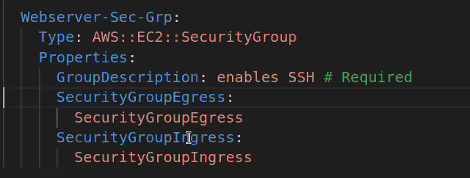
Securitygroup oluşturmak istersek VS Code da securirty group yazdığımızda extension bize bu taslağı getiriyor:



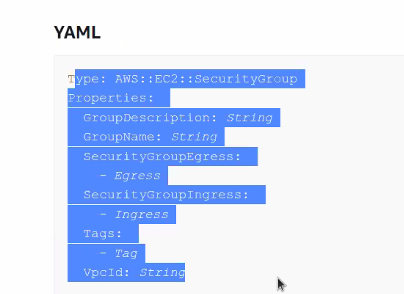


Bu resource da logicalid vermemiz gerekiyor ancak aynısını veremeyiz. Web-server-sec diyelim.

vermeyeceğimiz verilerin keylerini sildik:



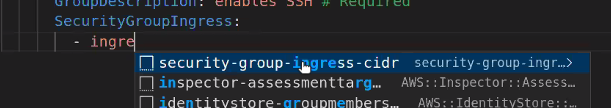
Engress outband rules demek. onu da default bıraktığımız için siliyoruz. Ingress için de dokumantasyona gidip sol menüden EC2 securitygroupu seçip ingresse geliyoruz ve syntax ine bakıyoruz.

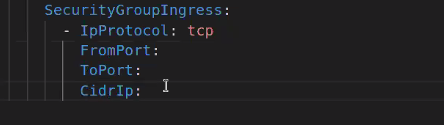


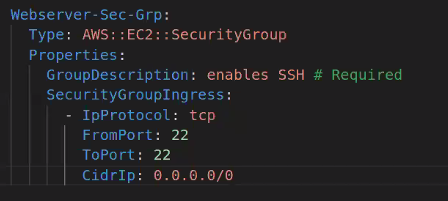
List şeklinde olacak syntaxi de:



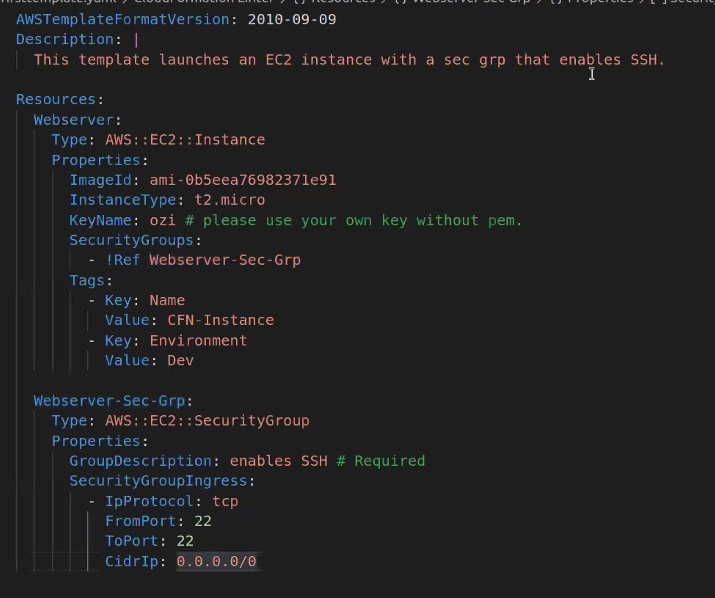
kopyalamaya gerek yok vs code extension u bize bunu verecek. list şeklinde yazıyoruz:





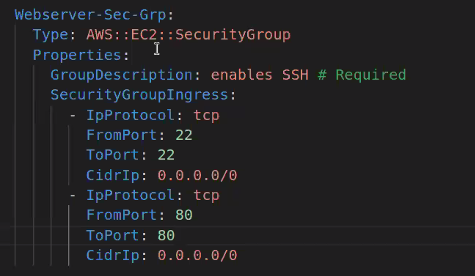


Biz burayı instance bloğunda referans göstermemiz gerekiyor:



üstte instance SecurityGorups altındaki list e !Ref diyerek aşağıdaki securiygroup id sini yazacaz. Bu şekilde referans vermiş olduk.

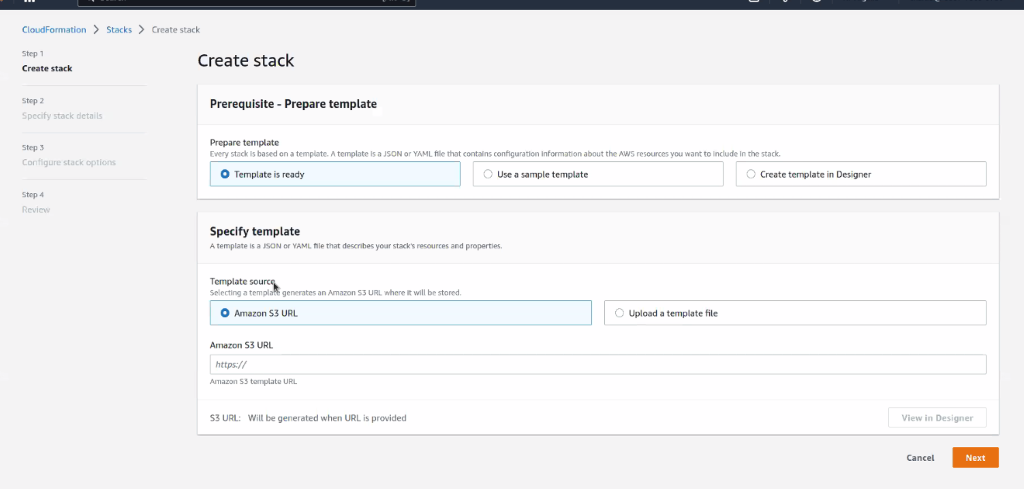
80 portunu da security group a eklemek için:



aynı verileri 80 olarak giriyoruz. Template hazır.

(Webserver-Sec-Grp ta hata verid tire - leri silince düzeldi)

CloudFormation sayfasına gidip Create Stack diyoruz:

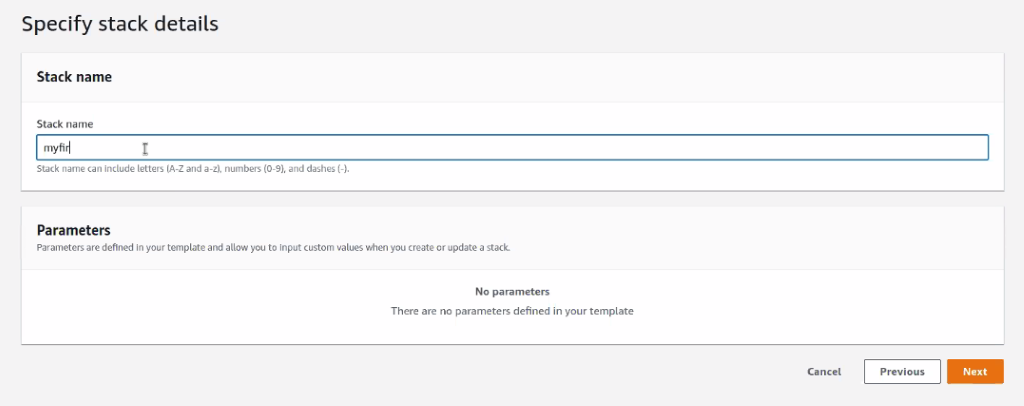


Template is ready diyoruz. Çünkü kendimiz hazırladık. S3 templatemiz olsaydı bucket takip url sini yazarak oradan almasını sağlayabilirdik.

Upload a template file diyerek lokaldeki dosyamızı seçiyoruz:

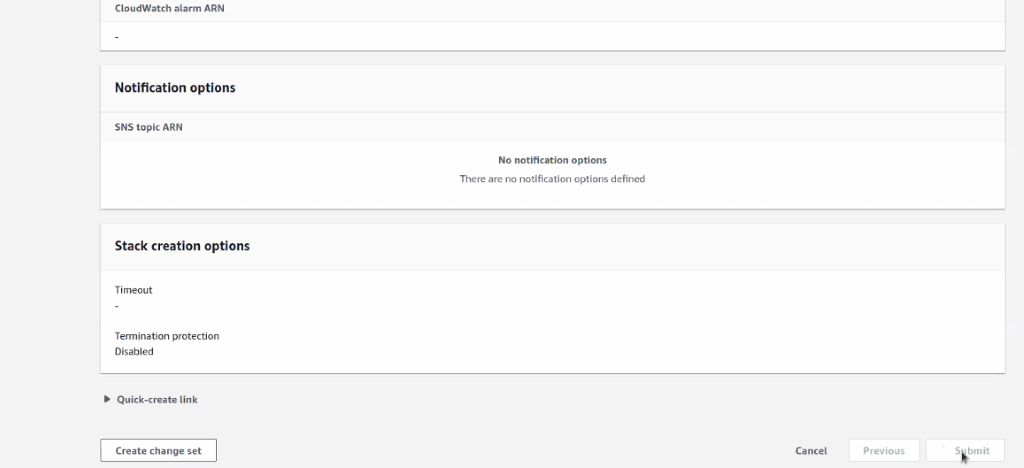


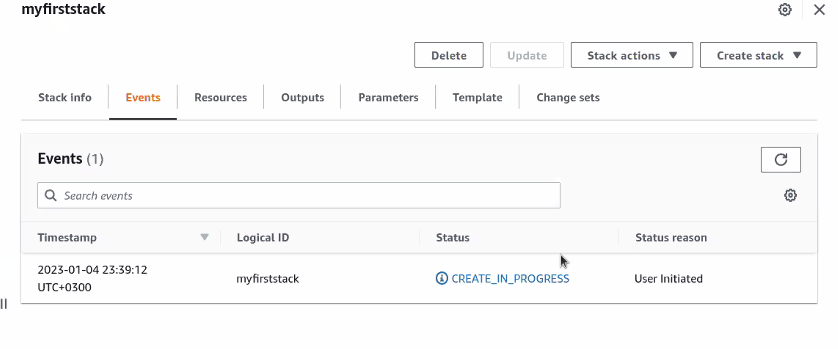
Next diyoruz:



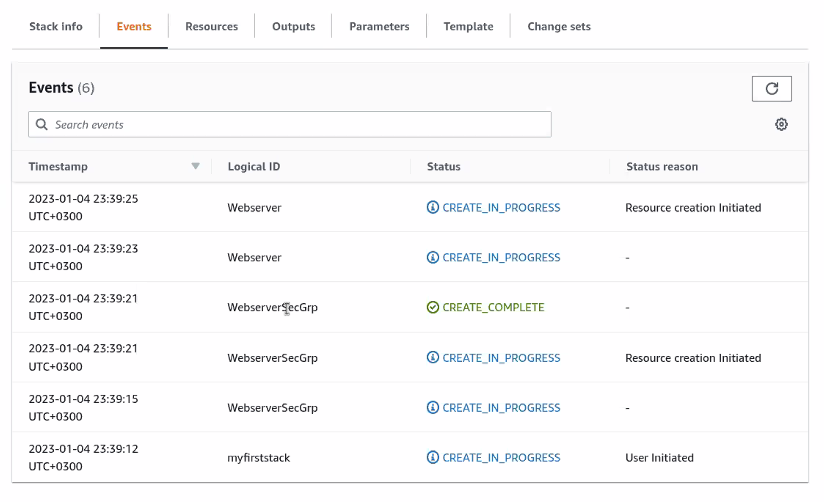
Burada isim veriyoruz. Template bloklarında parametre bölümü vardı. Parametre belirleseydik bu ekranda karşımıza çıkacaktı.

Next diyoruz. Sonraki ekranda da next diyoruz sonra submit diyoruz:

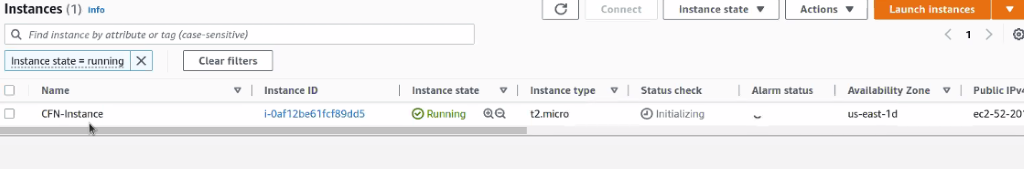




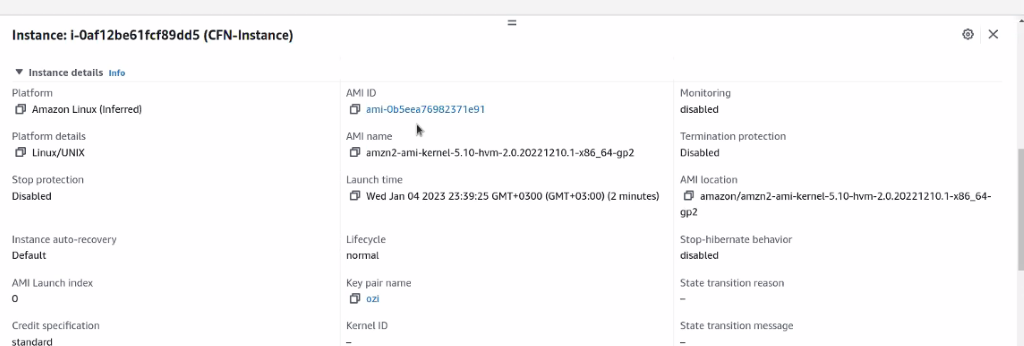
sağda status reason diyor, user initated kullanıcı tarafından oluşturuldu diyor. Events ta da işlem basamklarını gösteriyor:



Instances a giderek kontrol ettiğimizde ayağa kalktığını görüyoruz:



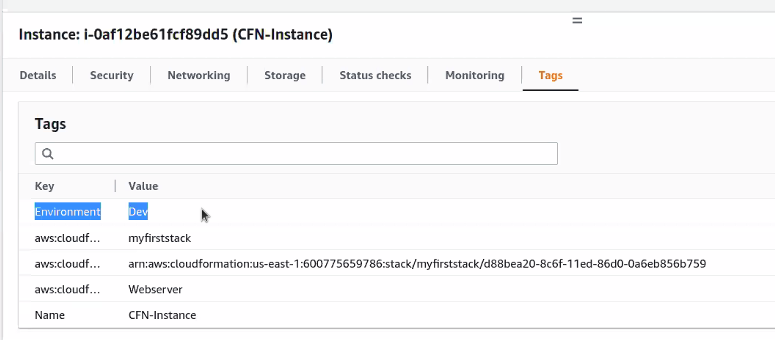
ami id miz ve key pair name verdiğimiz gibi:



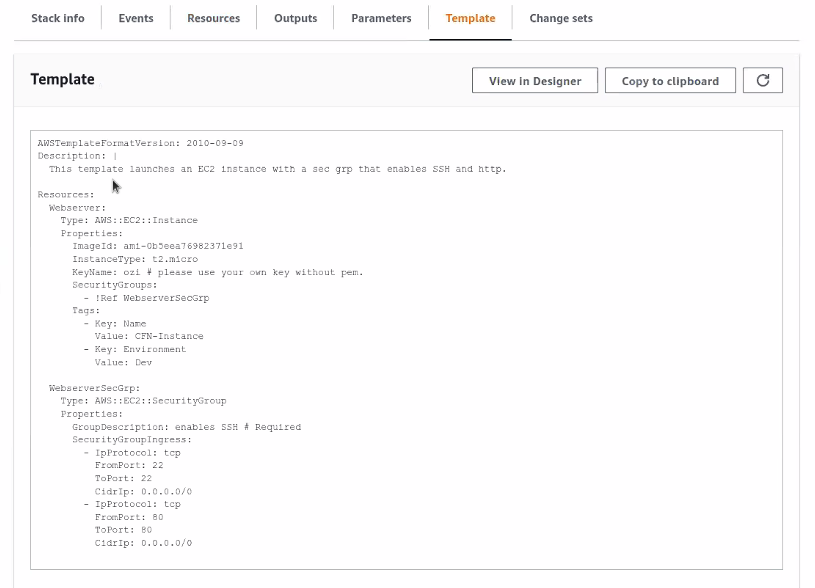
security group verdiğimiz id ile görünüyor:



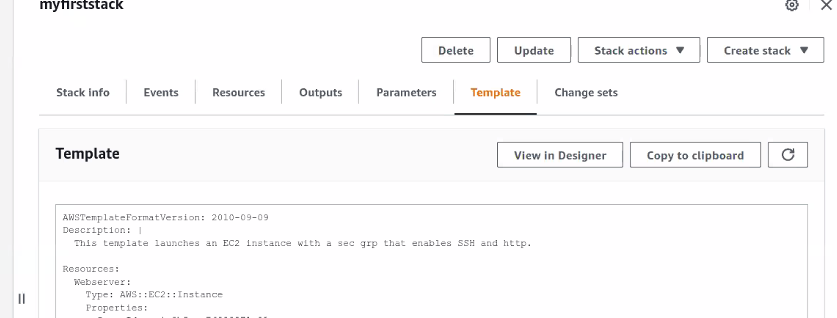
verdiğimiz tag ler görünüyor:



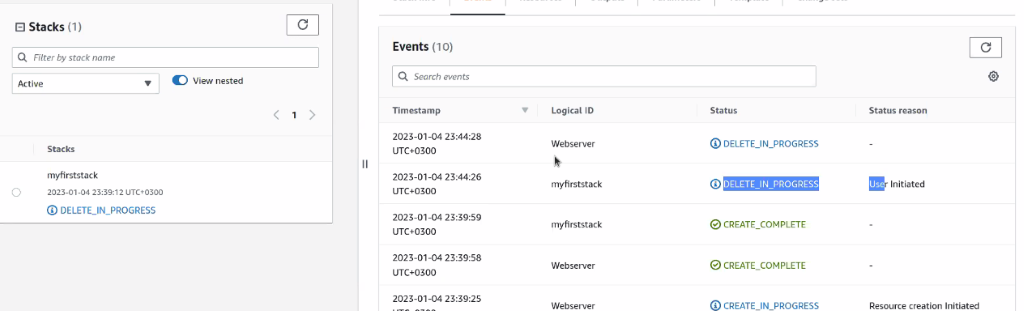
Template ler stack menüsünde görünüyor:



Delete Stack dediğimizde;



Events ten deleting progress i görüyoruz:



Delete ettiğimizde ayağa kaldırdığı tüm resource u siler.