

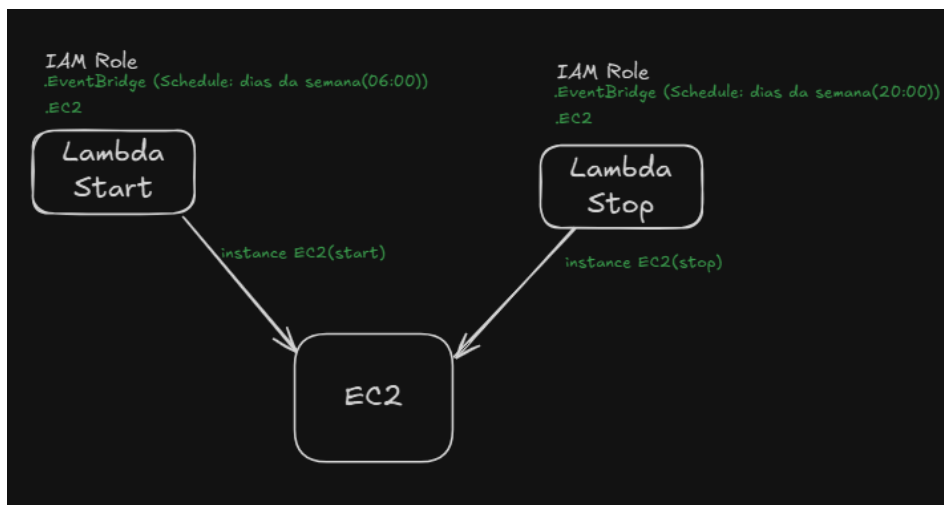
Start/Stop Maquinas EC2

Apresentação Projeto

Nesse projeto, iremos automatizar o **desligamento** e a **ligacao** de **maquinas EC2** em certos horarios nos dias da semana, visando a economia de recursos e financeiros!!

Pois pode ser que o consumo dessas maquinas **EC2** so seja necessario durante o horario comercial e durante a semana, logo precisaremos de **2 scripts!!** Um que realiza esse desligamento e outro que realiza a reativacao das maquinas!!

Time-Line Projeto



Criacao maquinas EC2

Primeiramente precisamos criar maquinas **EC2** para que os nossas **funcoes lambda** possam ligar e desligar as maquinas **EC2** dependendo do horario esperado!

Instâncias (1)		Informações	Última atualização less than a minute atrás	Conectar	Estado da instância	Ações	Executar instâncias
Localizar Instância por atributo ou tag (case-sensitive)							
Todos os ...							
<input type="checkbox"/>	Name	ID da instância	Estado da inst...	Tipo de inst...	Verificação de stal	Status do alarm	Zona de dispon...
<input type="checkbox"/>		i-0d8e11a253af5e6f2	Executando	t2.micro	Inicializando	Exibir alarmes	us-east-1a

Criando funcoes lambda

Com as maquinas **EC2** criadas, vamos criar agora as **funcoes lambda**, para podermos automatizar a o ligamento e o desligamento das maquinas **EC2!!**

Funções (2)

Última busca: há 10 segundos

Ações

Criar função

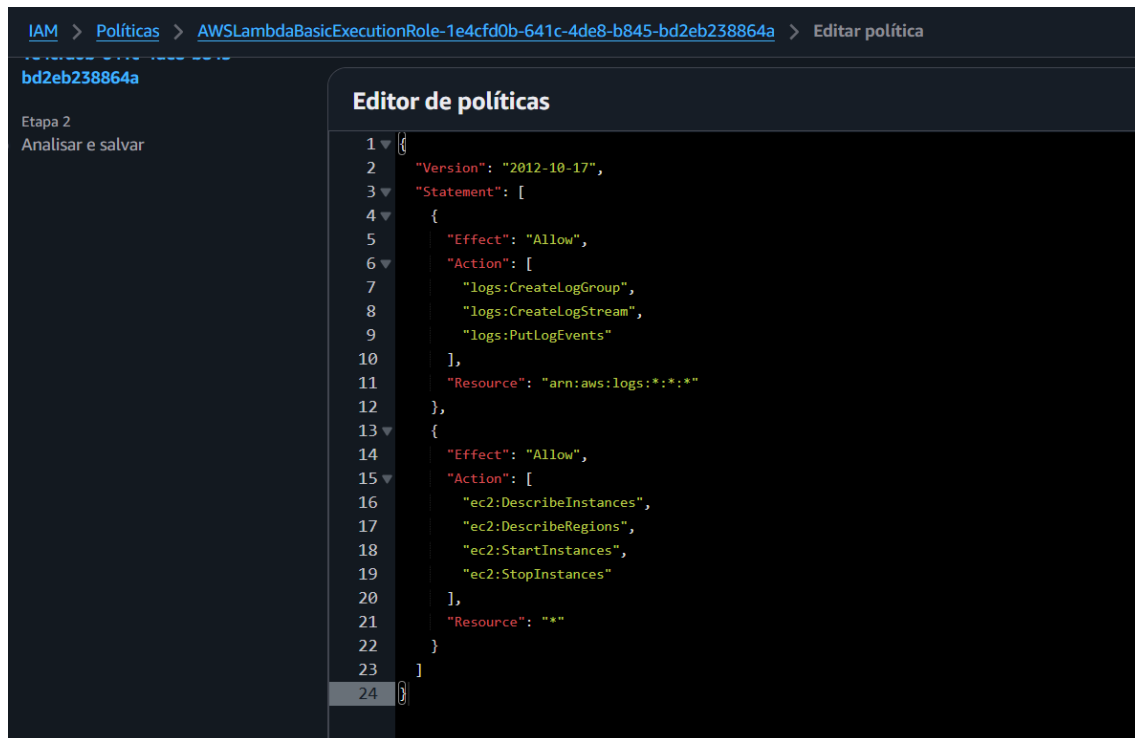
Filtrar por atributos ou pesquisar por palavra-chave

<input type="checkbox"/>	Nome da função	Descrição	Tipo de pacote	Tempo de execução	Última modificação
<input type="checkbox"/>	Start_EC2	-	Zip	Python 3.13	há 57 segundos
<input type="checkbox"/>	Stop_EC2	-	Zip	Python 3.13	há 8 segundos

Permissao funcoes lambda

Agora precisamos ajustar as regras de permissões do **IAM** de nossas **funcoes lambda**!!

Pois como ja sabemos por padrao a unica permissao que as **funcoes lambda** possui eh referente o servico do **cloudwatch**, e teremos que adicionar as permissões referente ao servico de **EC2**!! Como podemos ver a seguir:



Script(Python) - funcao lambda(Stop_EC2)

Vamos agora criar os nossos **scripts python** de nossas **funcoes lambda**, e para comecar vamos criar o **script** referente a **funcao lambda(Stop_EC2)**.

E para iniciar vamos logo importar a biblioteca do **boto3**, que como ja sabemos se trata da biblioteca(python) **oficial** da **aws** para integrar diretamente com os seus servicos!! E claro, a criacao da funcao **lambda_handler(event, context)**!!

```
1 lambda_function.py
2 import boto3
3
4 def lambda_handler(event, context):
5     pass
```

Em seguida, vamos instanciar o objeto **ec2_client** alem de listar todas as regioes possiveis de nossa conta aws, atraves pelo metodo **ec2_client.describe_regions()['Regions']**.

```
def lambda_handler(event, context):
    ec2_client = boto3.client('ec2')
    regions = [region['RegionName'] for region in ec2_client.describe_regions()['Regions']]
```

Veja que foi feito um **List Comprehension**, para que seja retornado uma nova lista alternativa, referente as **regioes** em si, que podemos ter **instancias EC2** criadas na nossa conta **aws**!

Feito isso, vamos criar um **for loop**, para que seja possivel iterar para cada regioao retornado no nosso **List Comprehension**!

```
for region in regions:
    pass
```

E como primeira etapa do nosso **for loop**, vamos utilizar funcao **boto3.resource()** da library do **boto3**, onde buscaremos as nossas **instancias EC2**!! E veja, que definimos a regioao especifica tambem que queremos buscar pelo parametro **region_name**!!

```
for region in regions:
    print("Region:", region)
    ec2 = boto3.resource('ec2', region_name=region)
```

Com as **instancias EC2** retornadas, precisamos agora filtrar apenas as **instancias** que estao com o status **RUNNING**, para que possamos realizar o **STOP**, e para isso utilizamos a prop **'instance-state-name'**!!

```
instances = ec2.instances.filter(
    Filters=[{'Name': 'instance-state-name', 'Values': ['running']}])
```

E para finalizar o script, vamos realizar um novo for loop para que de fato realizar o stop dessas instancias EC2

```
for instance in instances:
    instance.stop()
    print('Stopped instance: ', instance.id)
```

Veja que foi stopado atraves do metodo **stop()** pelo objeto **instance**!

Script(Python) - funcao lambda(Start_EC2)

Vamos agora criar o nosso **scripts python** de nossa **funcao lambda(Start_EC2)** e veja que o script eh praticamente o mesmo!! Com apenas essas alteracoes:

- **Filtro das intancias EC2:** agora filtramos a prop **'instance-state-name'** pelo valor **'stopped'**.
- **Metodo que inicializar a instancia EC2:** agora como a **instancia EC2** esta stopada, precisamos inicializar atraves do metodo **start()** pelo objeto **instance**.

```
lambda_function.py
1  import boto3
2
3  def lambda_handler(event, context):
4      ec2_client = boto3.client('ec2')
5      regions = [region['RegionName'] for region in ec2_client.describe_regions()['Regions']]
6
7      for region in regions:
8          print("Region:", region)
9
10         ec2 = boto3.resource('ec2', region_name=region)
11
12         instances = ec2.instances.filter(
13             filters=[{'Name': 'instance-state-name', 'Values': ['stopped']}])
14         )
15
16         for instance in instances:
17             instance.start()
18             print('Started instance: ', instance.id)
```

Configurando Schedule pelo Amazon EventBridge

E para finalizar, vamos agora configurar o **schedule(agendamento)** da execucao da nossa **funcao lambda!!** E para isso precisamos ir no servico do **EventBridge!!**

E para a criacao de nossas regras de agendamento, vamos utilizar **expressao cron!!** E como foi citado na **time-line do projeto**, para a **funcao lambda(Start_EC2)** sera feita todos os dias da semana as **06:00:00!** Ja para **funcao lambda(Stop_EC2)** sera feita todos os dias da semana as **20:00:00!**

- **Schedule funcao lambda(Start_EC2)**

The screenshot shows the 'Tipo de cronograma' (Schedule type) section in the Amazon EventBridge console. The 'Cronograma baseado em cron' (Cron-based schedule) option is selected. Below this, the 'Expressão cron' (Cron expression) is defined as 'cron (0 6 ? * MON-FRI)'. The '10 data de ativação próxima' (Next 10 activation dates) are listed as follows:

Data	Horário
Tue, 03 Jun 2025	06:00:00 (UTC-03:00)
Wed, 04 Jun 2025	06:00:00 (UTC-03:00)
Thu, 05 Jun 2025	06:00:00 (UTC-03:00)

- **Schedule funcao lambda(Stop_EC2)**

The screenshot shows the 'Tipo de cronograma' (Schedule type) section in the Amazon EventBridge console. The 'Cronograma baseado em cron' (Cron-based schedule) option is selected. Below this, the 'Expressão cron' (Cron expression) is defined as 'cron (0 20 ? * MON-FRI)'. The '10 data de ativação próxima' (Next 10 activation dates) are listed as follows:

Data	Horário
Mon, 02 Jun 2025	20:00:00 (UTC-03:00)
Tue, 03 Jun 2025	20:00:00 (UTC-03:00)
Wed, 04 Jun 2025	20:00:00 (UTC-03:00)

Cronogramas (2)

Desabilitar

Editar

Excluir

Criar cronograma

Pesquisar cronogramas carregados

Todos os estados

Todos os grupos

< 1 >

<input type="checkbox"/>	Nome do cronograma ▾	Grupo de programação ▾	Status ▾	Destino ▾	Tipo de destino ▾	Última modificação ▾
<input type="checkbox"/>	Schedule_EC2_Stop	default	<div><div></div>Habilitado</div>	Stop_EC2	LAMBDA_Invoke	Jun 02, 2025, 17:30:04 (UTC+00:00)
<input type="checkbox"/>	Schedule_EC2_Start	default	<div><div></div>Habilitado</div>	Start_EC2	LAMBDA_Invoke	Jun 02, 2025, 17:28:25 (UTC+00:00)

E pronto, com isso criamos todos os **schedules** necessarios para rodar as nossas **funcoes lambdas** referente ao horario correto!