Securely access AWS Parameter Store from

your Elastic Beanstalk Docker containers

19 June 2017 on aws, ec2, security, docker, iam, elasticbeanstalk

Reasoning

Managing configuration is always a tough choice. Whether you're following the 12-Factor methodology environment variables suggestion, classic configuration files or any other approach, the discussion of "how safe it is?" always comes up. AWS provides the **Parameter Store** service, inside the EC2 Systems Manager services. It is an easy to use configuration manager consisting of an account-wide store, a small API to do typical operations like get, put, list or delete configuration values, and integration with some of their services ecosystem, mainly IAM and CloudFormation.

We are going forward regarding using Elastic Beanstalk for at least one of our new projects regarding application lifecycle and deploy, placing our applications inside Docker containers. The why and how (and it's caveats) it's a topic we'll leave for future posts, but focusing on the subject at hand, we wanted our containers to **securely access Systems Manager Parameter Store and retrieve from there any configuration value**, whenever is a Redis hostname or a third-party API key.

Our Solution

After some research and experiments, we have a working solution, which we feel is not perfect, as it implies installing AWS CLI on the Docker instances, but it works and we keep adhering to our requirement of configuration via environment variables on the containers (and not on the host machine).

This is our current setup:

- EC2 **Systems Manager Parameter Store** parameters of type `SecureString` (encrypted with KMS).
- An **IAM policy,** restricted to:
  - Allow only `ssm:GetParameters` action (e.g. no listing of all parameters allowed).
  - Allow access to only parameters "namespaced" to our application. Example policy JSON:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ssm:GetParameters"
      ],
      "Resource": "arn:aws:ssm:eu-west-1:<account-id>:parameter/my-app.*"
    }
  ]
```

```
}
```

- An **IAM role** that uses the previous policy.
- **Elastic Beanstalk Docker container** application. Instances will either be `Amazon Linux running Docker` or `Ubuntu 16`, all with the custom IAM policy applied to them.
- **AWS CLI** installed on the containers, via `Dockerfile` or `requirements.txt`:

```
RUN pip install awscli
```

- **Custom shell script** that will run upon instance boot via `Dockerfile`, which dumps configuration parameters to environment variables:

```
ENTRYPOINT ["/tmp/start.sh"]
```

Contents of the shell script:

```
#!/usr/bin/env bash
set -e

export CONFIG_VALUE=$(aws ssm get-parameters --names
config-param --region eu-west-1 --with-decryption
--query Parameters[0].Value --output text)

exec python /tmp/application.py
```

Conclusions

With this solution, we achieve our main goal of **limiting exposure of configuration parameters** as much as possible (e.g. avoiding dumping them to a configuration file), while keeping our desired environment variables approach.

We could also use `boto3` (the official Python library for interacting with AWS) so there would be no intrinsic risk of having environment variables "alive" inside the containers, but until we have more services and experience with Elastic Beanstalk, we prefer the flexibility of env vars.

Miscellaneous notes

Amazon advises to install their **SSM Agent**, and we did so via `Dockerfile`:

```
WORKDIR /tmp
```

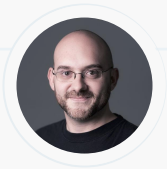
```
RUN wget https://s3.amazonaws.com/ec2-downloads-windows/SSMAgent/latest/debian_amd64/amazon-ssm-agent.deb
RUN dpkg -i amazon-ssm-agent.deb
```

But we tried removing the agent, and AWS CLI still works fine, which makes sense as IAM roles should be enough, and the agent seems to be more oriented to being able to execute remote commands and notify Systems Manager of instances inventories/fleets (which we don't use).

We did multiple tests and checked quite a few AWS documentation pages and internet articles containing different approaches. For example, one thing we tested was doing the environment variables setup/exposure on the Docker host instead of the machines, but mainly found that:

- Using `.ebextensions` with normal `command` environment variables are not propagated to containers.
- Using `.ebextensions` with `option_settings` and overrides via a `command` doesn't work and only default value is propagated to the containers.
- Using `.ebextensions` with special deploy folders and shell scripts like `/opt/elasticbeanstalk/hooks/appdeploy/post/export_env_vars_on_host.sh` is discouraged by AWS support. We do use `/appdeploy/pre/` hooks but only to do trivial operations like `Dockerfile` composing (from multiple pieces).

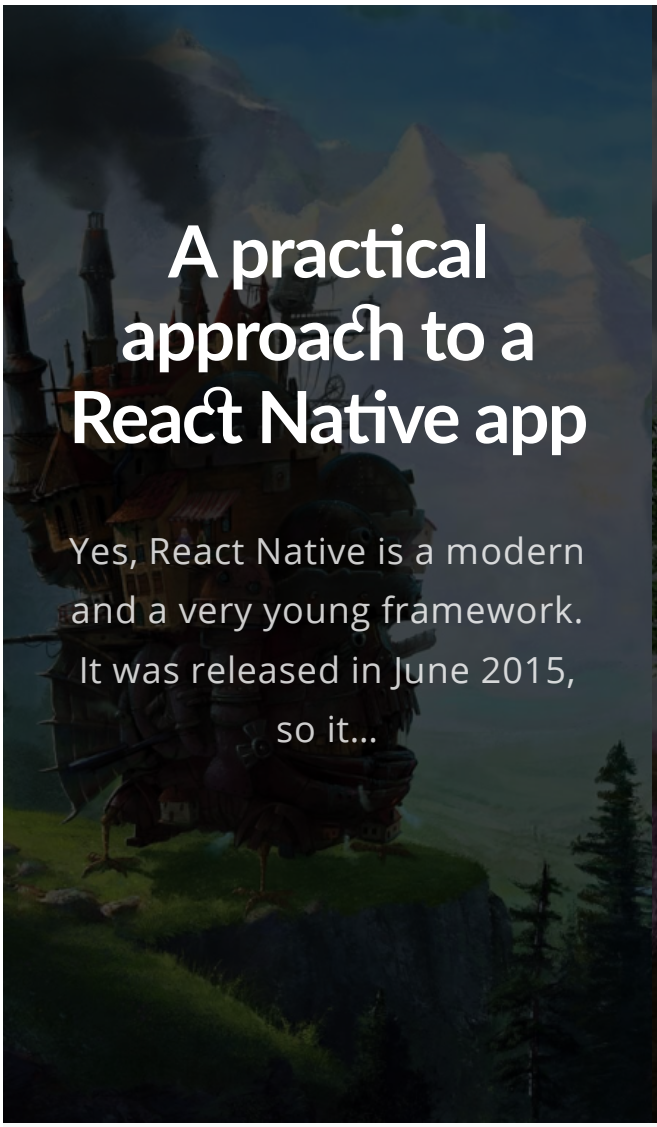
- Using `aws elasticbeanstalk update-environment` on the host to update itself the value of some variables forces us to give additional permissions to the IAM role and still feels a risky approach, so we discarded it.
- `{{ssm:<parameter-name>}}` Replacements on configuration files are not supported by Elastic Beanstalk, seems to be a feature only for EC2 Systems Manager's `Run Command`, and only supports alphanumeric parameter names (e.g. `myapp.prod.myvar` won't work).



Diego Muñoz 'Kartones'

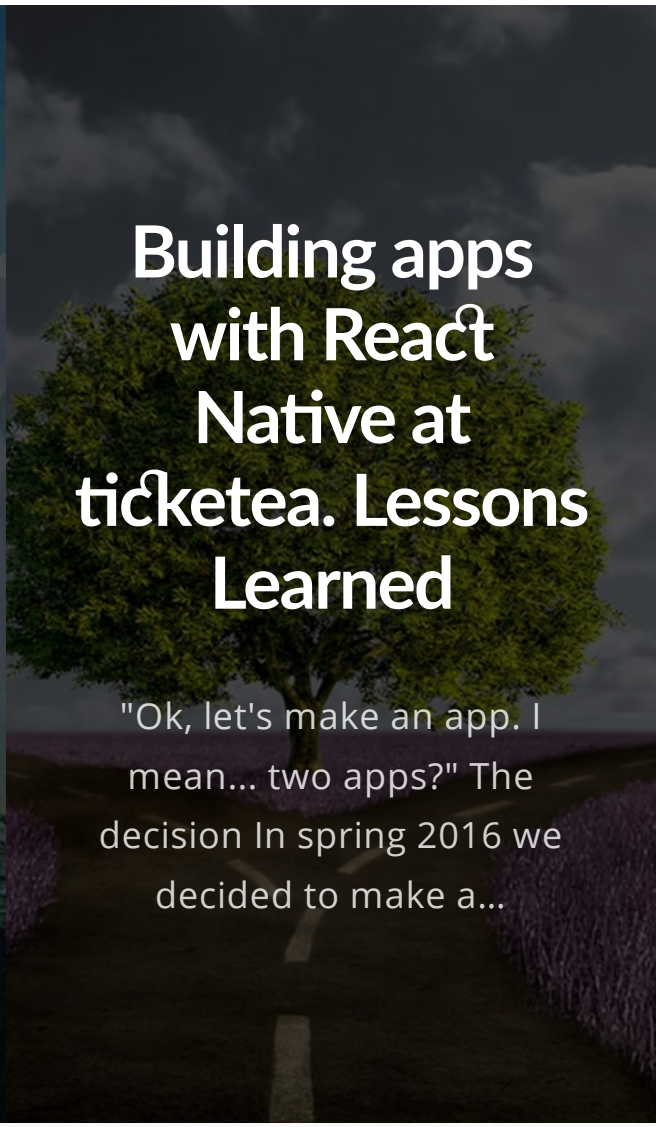
Read more posts by this author.

0.0.0.0, Madrid https://kartones.net



A practical approach to a React Native app

Yes, React Native is a modern and a very young framework. It was released in June 2015, so it...



Building apps with React Native at ticketea. Lessons Learned

"Ok, let's make an app. I mean... two apps?" The decision In spring 2016 we decided to make a...