**Monaco implementation**

Monaco is a CLI tool that automates the deployment of Dynatrace Monitoring Configuration to one or multiple Dynatrace environments.

The Monitoring as Code (Monaco) approach enables you to manage your Dynatrace environment monitoring tasks through configuration files instead of a graphical user interface. Configuration files allow you to create, update, and manage your monitoring configurations safely, consistently, and repetitively. They can be reused, versioned, and shared within your team.

* Having the ability to templatize our configuration for reusability across multiple environments
* Interdependencies between configurations should be handled without keeping track of unique identifiers
* Introducing the capability to easily apply – and update – the same configuration to hundreds of Dynatrace environments as well as being able to roll out to specific environments
* Identify an easy way to promote application specific configurations from one environment to another – following their deployments from development, to hardening to production.
* Support all the mechanisms and best-practices of git-based workflows such as pull requests, merging and approvals
* Configurations should be easily promoted from one environment to another following their deployment from development to hardening to production

Let’s look at how monitoring–as-code Tool (Monaco) fulfills these requirements:

1. Monaco speaks JSON  
   Simply placing a JSON object returned by our API into the right folder structure and adding a YAML config file, enables you to use the tool to setup that configuration on a Dynatrace environment – or thousands of environments.
2. Configuration as templates  
   The ability to ‘reuse anything’ as much as possible is a key requirement. For the same reason application developers create libraries for re-usable functionalities, we also see certain configurations coming back with different parameters. We might have a synthetic test template for a particular application which needs to be deployed in dev, staging and production. By reusing the same test template with different values for the endpoint, we massively reduce the effort to create and maintain these tests.
3. Referencing other configurations by name  
   Identifiers are great! They allow us to easily create a primary key of a configuration that is unique in the system. Unfortunately, if these get auto generated – which they are in most cases – they are hard to keep track of and reuse in other configurations. So, when we want to use a particular management zone in a dashboard, we wouldn’t want to have to look up that Management zone ID to reference it in our configuration as code. Monaco resolves this by giving you the ability to reference other configurations by name. Monaco will then look up the matching configuration using the Dynatrace API and provide the correct ID. Oh, and in case configurations depend on other configurations, Monaco will figure out in which order to deploy them automatically, to ensure all configs exist when they are needed. Piece of cake!
4. A GitOps approach to observability  
   Relying on JSON and YAML files for configuration allows for monitoring configuration to put under version control and changes to be reviewed and approved in the same way as code pull request. Using such versioned configuration, Monaco can be run as part of [CI/CD pipelines](https://www.dynatrace.com/news/blog/understanding-continuous-integration-and-continuous-delivery-ci-cd/) to ensure the right monitoring setup for any new deployment of the applications you’re monitoring.

This is how we use Monaco, git and a CI/CD pipeline at Dynatrace:

1. We have one repository that contains all our “main” configuration that needs to be applied to hundreds of environments.
2. Each time we need to make a change – e.g., add a new management zone – we create a new branch in this repository and create and test the new configuration.
3. When development and testing is done, a pull request is created to merge this new configuration into the main branch. This is then peer-reviewed for quality.
4. Once approved and merged, a pipeline kicks in that applies the configuration to all defined environments.

Steps of implementing in Monaco:-

./monaco deploy --project kde873 --environments=b2b\_test\_DT\_environments.yaml

./monaco download --environments=b2b\_test\_DT\_environments.yaml

export NEW\_CLI=1 monaco

export MONACO\_REQUEST\_LOG=request.log

export MONACO\_RESPONSE\_LOG=response.log

1. Project Name
2. Template
3. every template:-json,yaml file.
4. environment yaml
5. monaco executable file

|  |  |
| --- | --- |
| Configuration | Token Permissions |
| alerting-profile | Read Configuration & Write Configuration. |
| Anomaly-detection-metrics | Read Configuration & Write Configuration. |
| Anomaly-detection-disks | Read Configuration & Write Configuration. |
| Anomaly-detection-applications | Read Configuration & Write Configuration. |
| Anomaly-detection-database services | Read Configuration & Write Configuration. |
| Anomaly-detection-hosts | Read Configuration & Write Configuration. |
| Anomaly-detection-Metrics | Read Configuration & Write Configuration. |
| Anomaly-detection-Services | Read Configuration & Write Configuration. |
| App-detection-rule | Read Configuration & Write Configuration. |
| App-detection-rule-host | Read Configuration & Write Configuration. |
| Application | Read Configuration & Write Configuration. |
| auto-tag | Read Configuration & Write Configuration. |
| Conditional-naming | Read Configuration & Write Configuration. |
| custom-service | Read Configuration & Write Configuration. |
| dashboard | Read Configuration & Write Configuration. |
|  | Read Configuration & Write Configuration. |