Sloctl User Guide

sloctl, a command-line tool from Nobl9 to configure and manage SLO resources

Back to Nobl9 Documentation

The command-line interface (CLI) for Nobl9 is named `sloctl`. As a best practice, use the sloctl CLI when creating or updating multiple SLOs at once, creating or updating multiple objectives, or when updating SLOs as part of CI/CD. The web user interface is available to give you an easy way to create and update SLOs, and to familiarize you with the features available in Nobl9, while `sloctl` aims to provide a systematic and/or automated approach to maintaining SLOs as code.

Setting Up Sloctl

Use the following steps to install and configure Nobl9 command-line interface, sloctl.

Binary Installation

When installing files in protected folders, the operating system occasionally requires copy or file permissions. When this happens, give the installed files executable permissions (Linux and Mac) or confirm the file copy operation (Windows).

1. For **Mac**, we recommend using `brew`:
   - Install Homebrew
   - Use Terminal to complete the installation:
     ```
     brew tap nobl9/sloctl
     brew install sloctl
     ```
2. For **Windows** and **Linux**, we recommend manual installation from an archive:
   - Download the appropriate binary executable zip file from https://github.com/nobl9/sloctl/releases.
   - Extract the binary. Copy the executable file into the following platform-specific location:
     - **Linux**: `/usr/local/bin`
       > *For further details, see Linux instructions.*
     - **Windows**: `C:\Windows\System32`
       > *For further details and an example of installing into a separate program folder, see Windows instructions.*

Configuration

1. Create a **Client ID** and **Client Secret** pair for use in `sloctl`.
   - Navigate to **Settings** → **Access Keys** in the web UI.
   - Click **Create Access Key**.

2. Follow UI instructions to configure sloctl to use the provided credentials. Use one of the available set-up flows:
   - Click the **Download credentials file** in the web UI and put downloaded file in `~/.config/nobl9/config.toml` (Linux and macOS) or `%UserProfile%\.config\nobl9\config.toml` (Windows).

   or

   - Run `sloctl add-context`, name the context, and paste the **Client ID** and **Client Secret** from the web UI when prompted. Secrets will appear as `[hidden]` when using sloctl. Previously, secrets were excluded in the sloctl output.

3. Test the configuration by entering `sloctl get slos` into the command line.

   > 💡 *If there are no SLOs created in your account or in the selected project, you might see this message:* `No resources found in default project.`
   > *The message means the configuration is correct. The command line will return a* `401 error` *if the configuration does not work.*

General Usage

```
sloctl [command] [object] [--flags]
```

To get a list of all possible commands and flags, use one of the following forms:

```
sloctl -h
sloctl --help
```

To get details about a specific *command* and its objects and flags, use one of the following forms:

- `sloctl` *command* `-h`
- `sloctl` *command* `--help`

Command Summaries

The following are the available commands in sloctl.

All commands start with `sloctl`.
For example: `stoctl delete`

Command	Description
`add-context`	Add new sloctl configuration context.
`apply`	Apply resource definition in YAML or JSON format.
`delete`	Delete resource definition by name or definition file.

Command	Description
`get`	Display one or more than one resource.
`help`	Help about any command.
`use-context`	Set the default context.
`version`	Print the sloctl version.

Common Objects

These are the API objects that you can manipulate with `sloctl` with different commands. For details about each object, see the YAML Guide.

Objects arguments follow the command argument in the `sloctl` command line.
For example:

- `sloctl` *command object*
- `stoctl get agents`

Object	Description
`agents`	Provide a solution to metrics collection from external sources. In this solution users deploy the agents
`alertmethods`	When an incident is triggered, Nobl9 enables you to send the alert to a notification engine or tool
`alertpolicies`	A policy defining a set of conditions that when met, cause an alert to be a sent to a predefined list of integrations
`alerts`	Notifications for SLOs when certain conditions are met
`dataexports`	The configuration to export your data from Nobl9
`directs`	Provide a SaaS solution to metrics collection from external sources
`projects`	Serve as workspaces for resources and provide a layer of isolation for resources in different projects
`services`	High-level groupings of SLOs
`roleBindings`	Assign a user the permissions indicated in a role
`slos`	The set of target values for a service level objective

> ! ***Caution:*** *If you are using `sloctl` version older than 0.0.56, you will not be able to use the `kind: Project` and `kind: RoleBinding`.*

Flags

Common Flags pass data to a command or parameter.

Flag	Long Form	Description
`-h`	`--help`	Help for sloctl
`-o`	`--output` *string*	output format: one of `table|yaml|json` (default `yaml`)

> 💡 *Certain commands have other specific flags.*

Global Flags define the scope of the current command, such as project, context or location of the configuration file.

Flag	Long Form	Description
`-A`	`--all-projects`	Displays the objects from all of the projects.
	`--config` *string*	Config file path (without this flag checks for `%UserProfile%\.config\nobl9\config.toml` on Windows and `~/.config/nobl9/config.toml` on other operating systems).
`-c`	`--context` *string*	Overrides the default context for the duration of the selected command.
`-p`	`--project` *string*	Overrides the default project from active context for the duration of the selected command.

Command References

Help (Start screen)

```
sloctl
sloctl -h
sloctl --help
```

All available commands for execution are listed below. Use this tool to work with definitions of SLO in YAML files. For every command more detailed help is available.

Usage:

```
  sloctl [command]
```

Available Commands:

Command	Description
`add-context`	Add new sloctl configuration context.
`apply`	Apply resource definition in YAML or JSON format.
`delete`	Delete resource definition by name or definition file.
`get`	Display one or more than one resource.
`help`	Help about any command.
`use-context`	Set the default context.
`version`	Print the sloctl version.

Flags:

Flag	Long Form	Description

Flag	Long Form	Description
`-A`	`--all-projects`	Displays the objects from all of the projects.
	`--config` *string*	Config file path (without this flag checks for `%UserProfile%\.config\nobl9\config.toml` on Windows and `~/.config/nobl9/config.toml` on other operating systems).
`-c`	`--context` *string*	Overrides the default context for the duration of the selected command.
`-h`	`--help`	Help for `sloctl`
`-p`	`--project` *string*	Overrides the default project from active context for the duration of the selected command.

Use `sloctl [command] --help` for more information about a command.

Add-context

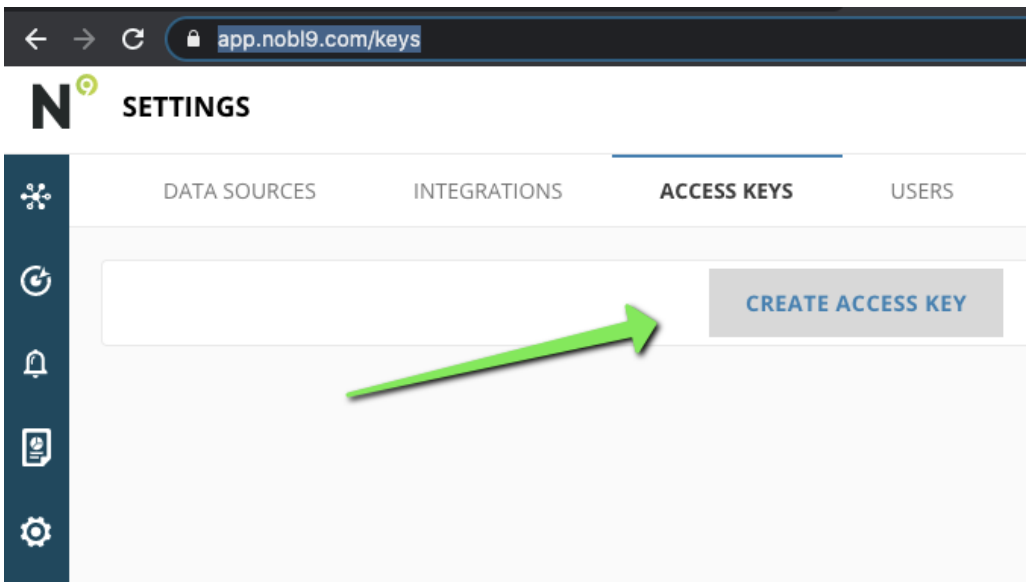
Add new sloctl configuration context, an interactive command, which collects parameters in wizard mode.

Usage:

```
sloctl add-context
```

1. Set the context name:
   `New context name:` *Enter the context name and press Enter*

2. Set the **Client ID**:
   In the Nobl9 application, the **Client ID** is generated by navigating to **Settings** -> **Access Keys** -> **Create Access Key**.



   - `Client ID:` *Enter the Client ID and press Enter*

3. Set **Client Secret**:

   - `Client Secret:` *Enter the Client Secret and press Enter*

4. Set a default **Project**:

   The project can be overridden with the `-p (--project)` flag in any command:

   - `Project [default]:` *Enter the default project name and press Enter*

5. You are asked if newly created context should be set as default now:

   - `Set '`*context name from Step 1*`' as a default context? [y/N]:` *type* `y` *(for Yes) or* `n` *(for No)*

   Note: hitting Enter without selecting `y` or `n` would default to *No*.

Use-context

You can define multiple contexts and `use-context` allows you to choose the default context. It is helpful if the user belongs to multiple organizations or if the user wants to switch between different default projects.

Usage

```
sloctl use-context context-name # non-interactive mode
sloctl use-context # interactive mode
```

When `sloctl use-context` is used in interactive mode and some contexts are already defined, the user is asked to select the context from the list.

For example,

- ```
  sloctl use-context
  Select the default context from the existing contexts [prod, staging, dev]:
  ```

  user types a name from the list of existing contexts

When no context is defined yet, an error is shown:

- ```
  Error: You don't have any contexts in the current configuration file.
  Add at least one context in the current configuration file and then set it as
  Run 'sloctl add-context' or indicate the path to the file using flag '--config
  ```

Version

Usage

```
sloctl version
```

The output includes product version and architecture. For example,

- `sloctl/0.0.50-HEAD-bc7ec082 (linux amd64 go1.16.3)`

Get

Prints a table of the most important information about the specified resources.

Usage:

```
sloctl get [object]
```

Available objects:

- `agents`, `alertmethods`, `alertpolicies`, `alerts`, `dataexports`, `directs`, `projects`, `rolebindings` , `services`, `slos`

For details, see Common Objects

Flags:

Flag	Long Form	Description
`-h`	`--help`	Help for sloctl
`-o`	`--output` *string*	output format: one of `table|yaml|json` (default `yaml`)

Global Flags:

Flag	Long Form	Description
`-A`	`--all-projects`	Displays the objects from all of the projects.
	`--config` *string*	Config file path (without this flag checks for `%UserProfile%\.config\nobl9\config.toml` on Windows and `~/.config/nobl9/config.toml` on other operating systems).
`-c`	`--context` *string*	Overrides the default context for the duration of the selected command.
`-p`	`--project` *string*	Overrides the default project from active context for the duration of the selected command.

Use `sloctl get [object] --help` for more information about the get command and different objects.

Notes:

- If there are no objects of given type associated with given project the user will see an information.

```
sloctl get slo

No resources found in 'default' project.
```

- If the user provides a valid type (i.e. slo) without name, a list of objects is returned.

```
sloctl get slo

- apiVersion: n9/v1alpha
  kind: SLO
  metadata:
    name: streaming-slo
    project: default
```

```
  spec:
    ...
- apiVersion: n9/v1alpha
  kind: SLO
  metadata:
    name: streaming-other-slo
    project: default
  spec:
    ...
```

- If the user provides an object type and name, only a specific object is returned.

```
sloctl get slo streaming-other-slo

- apiVersion: n9/v1alpha
  kind: SLO
  metadata:
    displayName: Streaming Other
    name: streaming-other-slo
    project: default
  spec:
    alertPolicies:
    - budget-is-burning-too-fast
    budgetingMethod: Occurrences
    description: ""
    indicator:
      metricSource:
        kind: Agent
        name: prometheus-source
        project: default
      rawMetric:
        prometheus:
          promql: cpu_usage_user{cpu="cpu-total"}
    objectives:
    - displayName: Good
      op: lte
      tag: default.streaming-other-slo.100d000000
      target: 0.9
      value: 100
    - displayName: Poor
      op: lte
      tag: default.streaming-other-slo.200d000000
      target: 0.99
      value: 200
    service: webapp-service
    timeWindows:
    - count: 7
      isRolling: true
      period:
        begin: "2021-04-28T13:09:35Z"
```

```
    end: "2021-05-05T13:09:35Z"
  unit: Day
```

- It is possible to pass multiple names, space separated.

```
sloctl get slo streaming-other-slo streaming-latency-slo
```

Delete

Delete different resources.

Usage:

```
sloctl delete [flags]
sloctl delete [command]
```

Examples:

- Delete the configuration from slo.yaml

```
sloctl delete -f ./slo.yaml
```

- Delete resources from multiple different sources at once.

```
sloctl delete -f ./slo.yaml -f test/config.yaml -f https://my-definition.loca
```

- Delete the YAML or JSON passed directly into stdin.

```
cat slo.yaml | sloctl delete -f -
```

- Delete by passing one or more resource names

```
sloctl delete slo my-slo-name
```

- Delete the configuration from slo.yaml and set project context if it is not defined in file

```
sloctl delete -f ./slo.yaml -p slo
```

Available objects:

- `agents`, `alermethods` , `alertpolicies`, `dataexports`, `directs`, `services`, `slos`

For details, see Common Objects

Flags:

Flag	Long Form	Description
`-f`	`--file` *[]string*	Provides a file path or a URL to the configuration in YAML or JSON format. This option can be used multiple times.
`-h`	`--help`	Help for sloctl.

Global Flags:

Flag	Long Form	Description
-A	--all-projects	Displays the objects from all of the projects.
	--config *string*	Config file path (without this flag checks for %UserProfile%\.config\nobl9\config.toml on Windows and ~/.config/nobl9/config.toml on other operating systems).
-c	--context *string*	Overrides the default context for the duration of the selected command.
-p	--project *string*	Overrides the default project from active context for the duration of the selected command.

Use `sloctl delete [object] --help` for more information about the delete command and different objects.

When the specified objects were deleted successfully, a confirmation message is shown:

```
sloctl delete -f example.yaml

The resources were successfully deleted.
```

Apply

The apply command commits the changes by sending the updates to the application.

Usage:

```
  sloctl apply [flags]
```

Examples:

- Apply the configuration from slo.yaml
  ```
  sloctl apply -f ./slo.yaml
  ```
- Apply resources from multiple different sources at once.
  ```
  sloctl apply -f ./slo.yaml -f test/config.yaml -f https://my-definition.com/s
  ```
- Apply the YAML or JSON passed directly into stdin.
  ```
  cat slo.yaml | sloctl apply -f -
  ```
- Apply the configuration from slo.yaml and set project if it is not defined in file
  ```
  sloctl apply -f ./slo.yaml -p slo
  ```

Flags:

Flag	Long Form	Description

Flag	Long Form	Description
`-f`	`--file` *[]string*	Provides a file path or a URL to the configuration in YAML or JSON format. This option can be used multiple times.
`-h`	`--help`	Help for sloctl

Global Flags:

Flag	Long Form	Description
`-A`	`--all-projects`	Displays the objects from all of the projects.
	`--config` *string*	Config file path (without this flag checks for `%UserProfile%\.config\nobl9\config.toml` on Windows and `~/.config/nobl9/config.toml` on other operating systems).
`-c`	`--context` *string*	Overrides the default context for the duration of the selected command.
`-p`	`--project` *string*	Overrides the default project from active context for the duration of the selected command.

Use `sloctl apply [object] --help` for more information about the delete command and different objects.

When the specified changes were applied successfully, a confirmation message is shown:

```
sloctl apply -f ./samples/sample.yaml

Resources successfully created.
```

Common Errors / Warnings

The following are common errors / warnings that users can experience.

- When user wants to use not existing context:

  ```
  sloctl use-context asdasd

  Error: There is no such context: 'asdasd'.  Please enter the correct name.
  ```

- When user wants to add context that already exists:

  ```
  sloctl add-context
  New context name: local

  Context 'local' is already in the configuration file.
  Do you want to overwrite it? [y/N]:
  ```

  If user answers `n` (for No), the following message is shown:

  ```
  Please try to add a new context with a different name.
  ```

- When context is configured with incorrect credentials:

  (config.toml context)

  ```
  defaultContext = "local"

  [Contexts]
     [Contexts.local]
         clientId = "xyz"
         clientSecret = "xyz"
  ```

  And the user wants to invoke any command, the following error is shown:

  ```
  Error: error getting new access token from the customer identity provider: ca
  ```

- When user provides path to invalid file:

  ```
  sloctl apply -f xyz.yaml

  Error: error while reading provided file: open xyz.yaml: no such file or direc
  ```

- When user provides invalid context name:

  ```
  sloctl add-context
  New context name: ^

  Error: Enter a valid context name. Use letters, numbers, and `-` characters.
  ```

Use Cases of SLO Configurations

The following examples explain how to create SLOs for sample services using `sloctl`.

A Typical Example of a Latency SLO for a RESTful Service

First, we want to pick an appropriate service level indicator to measure the latency of response from a RESTful service. In this example, let's assume our service runs in NGINX web server, and we're going to use a threshold-based approach to define acceptable behavior. For example, we want the service to respond in a certain amount of time.

> 💡 ***Note:*** *There are many ways to measure application performance. In this case we're giving an example of server-side measurement at the application layer (NGINX) but it might be advantageous for your application to measure this differently. For example, you might choose to measure performance at the client, or at the load balancer, or somewhere else. Your choice depends on what you are trying to measure or improve, as well as what data is currently available as usable metrics for the SLI.*

The threshold approach uses a single query, and we set thresholds or breaking points on the results from that query to define the boundaries of acceptable behavior. In the SLO YAML, we specify the indicator like this:

```
  indicator:
    metricSource:
      name: devlab-prometheus
```

```
      project: default
      kind: Agent
    rawMetric:
      prometheus:
        promql: server_requestMsec{job="nginx"}
```

In this example we use Prometheus. The concepts are similar for other metrics stores. We recommend running the query against your Prometheus instance and reviewing the result data, so you can verify that the query returns what you expect, and so that you understand the units: whether it's returning latencies as milliseconds or fractions of a second, for example. In this example we'll assume that the query returns values between 60 and 150 milliseconds with some occasional outliers.

Choosing a Time Window

We need to choose whether we want a rolling or calendar-aligned window.

- Calendar-aligned windows are best suited for SLOs that are intended to map to business metrics that are measured on calendar-aligned basis, such as every calendar month, or every quarter.
- Rolling windows are better for tracking "recent" user experience of a service.

  In the rolling time window setting, the support for sparse metrics with points that have timestamps consistently over one minute apart is limited. Nobl9 reserves the right to ignore sparse points (>1 min apart) that don't have a round minute timestamp.

For our RESTful service, we'll be using the Rolling window SLO primarily to measure recent user experience and to make decisions about the risk of changes, releases, and how best to invest our engineering resources on a week-to-week or sprint-to-sprint basis. We want the "recent" period that we're measuring to trail back long enough that our users would consider it recent behavior. We choose to go with a 28-day window, which has the advantage of containing an equal number of weekend days and weekdays as it rolls.

```
timeWindows:
  - count: 28
    isRolling: true
    unit: Day
```

Choosing a Budgeting Method

There are two budgeting methods to choose from: **Time Slices** and **Occurrences**.

Time Slices
In the Time Slices method, what we count (objective we measure) is how many good minutes were achieved (minutes where our system is operating within defined boundaries), compared to the total minutes in the window.

This is useful for some scenarios, but it has a disadvantage when we're looking at "recent user experience" as we are with this SLO. The disadvantage is that a bad minute that occurs during a low-traffic period (say, in the middle of the night for most of our users, when they are unlikely to even notice a performance issue) would penalize the SLO the same amount as a bad minute during peak traffic times.

In the Time Slices method, the support for sparse metrics with points that have timestamps consistently over one minute apart is limited. Nobl9 reserves the right to ignore sparse points (>1 min apart) that don't

have a round minute timestamp.

Occurrences
The Occurrences method is well suited to this situation. Occurrences count good attempts (in this example, requests that are within defined boundaries) against the count of all attempts (this means all requests, including requests that perform outside of defined boundaries). Since total attempts are fewer during low-traffic periods, it automatically adjusts to lower traffic volume.

```
    budgetingMethod: Occurrences
```

Establishing Objectives

In this example we've talked to our product and support teams and can establish the following objectives:

- The service has to respond fast enough that users don't see any lag in the web applications that use this service
- Our Product Manager thinks that 100ms (1/10th of a second) is a reasonable threshold for what qualifies at Okay latency. We want to try hit that 95% of the time, so we code the first objective like this:

```
    - target: 0.95
      displayName: Laggy
      value: 100
      op: lte
```

This objective requires that 95% of requests complete within 100ms.

You can name each objective however you want. We recommend naming them how a user of the service (or how another service that uses this service) might describe the experience at a given objective. Typically, we use names that are descriptive adjectives of the experience when the objective is not met. When the objective is violated, we can say that the user's experience is "Laggy".

- Some requests fall outside of that 100ms range. We want to make an allowance for that, but we also want to set other objectives so that we know that even in its worst moments, our service is performing acceptably, and/or that its worst moments are brief.

Let's define another objective. In the above objective, we allow 5% of requests to run longer than 100ms. We want most of that 5%, say 80% of the remaining 5% of the queries to still return within 1/4th of a second (250ms). That means 99% of the queries return within 250ms (95% +4%). Add an objective like this:

```
    - target: 0.99
      displayName: Slow
      value: 250
      op: lte
```

This objective requires that 99% of requests complete within 250ms.

- While that covers the bulk of requests, even within the 1% of requests that we allow to exceed 250ms, the vast majority of them should complete within half a second (500ms). Even within the 1% of requests that we allow to exceed 250ms, we want to make sure the vast majority of them complete within half a second (500ms). Add an objective like this:

```
    - target: 0.999
      displayName: Painful
      value: 500
      op: lte
```

This objective requires that 99.9% of requests complete within 500ms. Putting it all together, our SLO definition for the described use cases looks like this:

```
- apiVersion: n9/v1alpha
  kind: SLO
  metadata:
    name: adminpageload
  spec:
    service: devlab-service
    budgetingMethod: Occurrences
    indicator:
      metricSource:
        name: devlab-prometheus
        project: default
        kind: Agent
      rawMetric:
        prometheus:
          promql: server_requestMsec{job="nginx"}

    objectives:
      - target: 0.95
        displayName: Laggy
        value: 100
        op: lte

      - target: 0.99
        displayName: Slow
        value: 250
        op: lte

      - target: 0.999
        displayName: Painful
        value: 500
        op: lte

    timeWindows:
      - count: 28
        isRolling: true
        unit: Day
```

Deperecated

- `kind: Integration`:

With the 0.0.58 release `kind: Integration` has been deprecated and `kind: AlertMethod` has been introduced instead.

During the transition period users can still apply their YAML files that use `kind: Integration`. Nobl9 will no longer return it, but it will return `AlertMethod`. The `sloctl get` command will not return any results.
