L08 Early Detection of Misconfiguration

Markus Raab

Institute of Information Systems Engineering, TU Wien

19.05.2021

Points in Time

Learning Outcomes

Students will be able to

- recall points of time relevant in configuration management.
- remind some arguments about pull vs. push.
- remember various strategies for earlier reduction of misconfiguration.

When are settings used?

From the application's perspective:

Implementation-time: Configuration accesses are hard-coded in the source code. For example, architectural decisions [1] lead to implementation-time settings.

When are settings used?

From the application's perspective:

Implementation-time:	Configuration accesses are hard-coded in the source code. For example, architectural decisions [1] lead to implementation-time settings.
Compile-time:	Configuration accesses are resolved by the build system while compiling.

When are settings used?

From the application's perspective:

Implementation-time:	Configuration accesses are hard-coded in the source code. For example, architectural decisions [1] lead to implementation-time settings.
Compile-time:	Configuration accesses are resolved by the build system while compiling.
Deployment-time:	Configuration accesses are while the software is installed.

When are settings used?

From the application's perspective:

Implementation-time:	Configuration accesses are hard-coded in the source code. For example, architectural decisions [1] lead to implementation-time settings.
Compile-time:	Configuration accesses are resolved by the build system while compiling.
Deployment-time:	Configuration accesses are while the software is installed.
Load-time:	Configuration accesses are during the start of applications.

When are settings used?

From the application's perspective:

Implementation-time:	Configuration accesses are hard-coded in the source code. For example, architectural decisions [1] lead to implementation-time settings.
Compile-time:	Configuration accesses are resolved by the build system while compiling.
Deployment-time:	Configuration accesses are while the software is installed.
Load-time:	Configuration accesses are during the start of applications.
Run-time:	Configuration accesses are during execution after the startup procedure.

Detection of Misconfiguration

Viewpoint

Different viewpoint: now from configuration management perspective.

Phases when we can detect misconfigurations:

- Compilation stage in configuration management tool

Detection of Misconfiguration

Viewpoint

Different viewpoint: now from configuration management perspective.

Phases when we can detect misconfigurations:

- Compilation stage in configuration management tool
- Writing configuration settings on nodes

Detection of Misconfiguration

Viewpoint

Different viewpoint: now from configuration management perspective.

Phases when we can detect misconfigurations:

- Compilation stage in configuration management tool
- Writing configuration settings on nodes
- Starting applications (load-time)

Detection of Misconfiguration

Viewpoint

Different viewpoint: now from configuration management perspective.

Phases when we can detect misconfigurations:

- Compilation stage in configuration management tool
- Writing configuration settings on nodes
- Starting applications (load-time)
- When configuration setting is actually used (run-time)
  $\rightarrow$ Latent Misconfiguration

Detection of Misconfiguration

Viewpoint

Different viewpoint: now from configuration management perspective.

Phases when we can detect misconfigurations:

- Compilation stage in configuration management tool
- Writing configuration settings on nodes
- Starting applications (load-time)
- When configuration setting is actually used (run-time)
  $\rightarrow$ Latent Misconfiguration

Problem

Earlier versus more context.

L08 Early Detection of Misconfiguration

Markus Raab

Institute of Information Systems Engineering, TU Wien

19.05.2021

Push vs. Pull

Push vs. Pull

- Push is more interactive.

Push vs. Pull

- Push is more interactive.
- Push cannot do its job if nodes are not reachable.

Push vs. Pull

- Push is more interactive.
- Push cannot do its job if nodes are not reachable.
- Push needs additional techniques to scale with many nodes.

Push vs. Pull

- Push is more interactive.
- Push cannot do its job if nodes are not reachable.
- Push needs additional techniques to scale with many nodes.
- Push demands access to servers from a single server.

Push vs. Pull

- Push is more interactive.
- Push cannot do its job if nodes are not reachable.
- Push needs additional techniques to scale with many nodes.
- Push demands access to servers from a single server.
- Pull needs additional monitoring to know when a patch has been applied.

Push vs. Pull

- Push is more interactive.
- Push cannot do its job if nodes are not reachable.
- Push needs additional techniques to scale with many nodes.
- Push demands access to servers from a single server.
- Pull needs additional monitoring to know when a patch has been applied.
- Pull needs resources even if nothing is to do.

Push vs. Pull

- Push is more interactive.
- Push cannot do its job if nodes are not reachable.
- Push needs additional techniques to scale with many nodes.
- Push demands access to servers from a single server.
- Pull needs additional monitoring to know when a patch has been applied.
- Pull needs resources even if nothing is to do.

Task

Do you prefer push or pull? What does your CM tool of choice use?

L08 Early Detection of Misconfiguration

Markus Raab

Institute of Information Systems Engineering, TU Wien

19.05.2021

Early Detection

As shown by Xu et al. [2]:

- 12 % – 39 % configuration settings are not used at all during the application's startup procedure.

As shown by Xu et al. [2]:

- 12 % – 39 % configuration settings are not used at all during the application's startup procedure.
- Applications often have latent misconfigurations (14 % – 93 %).

As shown by Xu et al. [2]:

- 12 % – 39 % configuration settings are not used at all during the application's startup procedure.
- Applications often have latent misconfigurations (14 % – 93 %).
- Latent misconfigurations are particular severe (75 % of high-severity misconfigurations).

As shown by Xu et al. [2]:

- 12 % – 39 % configuration settings are not used at all during the application's startup procedure.
- Applications often have latent misconfigurations (14 % – 93 %).
- Latent misconfigurations are particular severe (75 % of high-severity misconfigurations).
- Latent misconfiguration needs longer to diagnose.

Checkers as plugins

Using checkers as plugins exclude whole classes of errors such as:

- Invalid file paths using the plugin *“path”*.

Checkers as plugins

Using checkers as plugins exclude whole classes of errors such as:

- Invalid file paths using the plugin *“path”*.
- Invalid IP addresses or host names using the plugins *“network”* or *“ipaddr”*.

Checkers as plugins

Using checkers as plugins exclude whole classes of errors such as:

- Invalid file paths using the plugin *"path"*.
- Invalid IP addresses or host names using the plugins *"network"* or *"ipaddr"*.

Because the checks occur before the resources are actually used, the checks are subject to race conditions.[1]

[1] For example, a path that was present during the check, can have been removed when the application tries to access it.

Checkers as plugins

Using checkers as plugins exclude whole classes of errors such as:

- Invalid file paths using the plugin *"path"*.
- Invalid IP addresses or host names using the plugins *"network"* or *"ipaddr"*.

Because the checks occur before the resources are actually used, the checks are subject to race conditions.[1]

In some situations facilities of the operating system help[2], in others we have fundamental problems.[3]

[1]For example, a path that was present during the check, can have been removed when the application tries to access it.

[2]For example, we open the file during the check and pass `/proc/<pid>/fd/<fd>` to the application. This file cannot be unlinked, but unfortunately the file descriptor requires resources.

[3]For example, if the host we want to reach has gone offline after validation.

Example [2]

Squid uses `diskd_program` but not before requests are served. Latent misconfiguration caused 7h downtime and 48h diagnosis effort.

Example [2]

Squid uses `diskd_program` but not before requests are served. Latent misconfiguration caused 7h downtime and 48h diagnosis effort.

Finding

Configuration from all externals programs need to be checked, too.

Conclusion

- provide external specifications for other tooling and configuration management

Conclusion

- provide external specifications for other tooling and configuration management
- use code generation to keep internal specifications consistent with external specifications (e.g. for refactoring)

Conclusion

- provide external specifications for other tooling and configuration management
- use code generation to keep internal specifications consistent with external specifications (e.g. for refactoring)
- implement checkers as plugins

Conclusion

- provide external specifications for other tooling and configuration management
- use code generation to keep internal specifications consistent with external specifications (e.g. for refactoring)
- implement checkers as plugins
- execute checkers as early as possible, also for external programs executed later

Conclusion

- provide external specifications for other tooling and configuration management
- use code generation to keep internal specifications consistent with external specifications (e.g. for refactoring)
- implement checkers as plugins
- execute checkers as early as possible, also for external programs executed later
- keep important resources allocated after checking

L08 Early Detection of Misconfiguration

Markus Raab

Institute of Information Systems Engineering, TU Wien

19.05.2021

Meeting

When are settings used?

From the application's perspective:

When are settings used?

From the application's perspective:

Implementation-time:	Configuration accesses are hard-coded in the source code. For example, architectural decisions [1] lead to implementation-time settings.
Compile-time:	Configuration accesses are resolved by the build system while compiling.
Deployment-time:	Configuration accesses are while the software is installed.
Load-time:	Configuration accesses are during the start of applications.
Run-time:	Configuration accesses are during execution after the startup procedure.

Application vs. CM tool perspective?

Application vs. CM tool perspective?

Phases when we can detect misconfigurations:

- Compilation stage in configuration management tool
- Writing configuration settings on nodes
- Starting applications (load-time)
- When configuration setting is actually used (run-time)
  $\rightarrow$ Latent Misconfiguration

Recapitulation

Task

Break.

Push vs. Pull

Push vs. Pull

- Push is more interactive.
- Push cannot do its job if nodes are not reachable.
- Push needs additional techniques to scale with many nodes.
- Push demands access to servers from a single server.
- Pull needs additional monitoring to know when a patch has been applied.
- Pull needs resources even if nothing is to do.

Task

Do you prefer push or pull? What does your CM tool of choice use?

Task

Break.

Talks

```
<talk>
  <date>19.05.2021</date>
  <name>@aaronabebe</name>
  <topic>externalized configuration in distributed systems</
</talk>

<talk>
  <date>19.05.2021</date>
  <name>@philippoppel</name>
  <topic>Configuration Integration based on T2</topic>
</talk>
```

Assignments

Please add slides for talk in TUWEL **and** private git repo, dates:

- 26 May: peer review

Assignments

T3 deadline today:
What means “only partial changes”?

Feedback

- ECTS breakdown realistic?
- Feedback Talk

Talks

```
<talk>
  <date>26.05.2021</date>
  <name>@a-kraschitzer</name>
  <topic>configuration migration</topic>
</talk>

<talk>
  <date>26.05.2021</date>
  <name>@robaerd</name>
  <topic>infrastructure as code</topic>
</talk>

<talk>
  <date>26.05.2021</date>
  <name>@tucek</name>
  <topic>A short introduction on how we configure our services at
</talk>
```

[1] Neil B Harrison, Paris Avgeriou, and Uwe Zdun. Using patterns to capture architectural decisions. *Software, IEEE*, 24(4):38–45, 2007. ISSN 0740-7459. doi: 10.1109/MS.2007.124.

[2] Tianyin Xu, Xinxin Jin, Peng Huang, Yuanyuan Zhou, Shan Lu, Long Jin, and Shankar Pasupathy. Early Detection of Configuration Errors to Reduce Failure Damage. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16)*, Savannah, GA, USA, November 2016.