Kamailio® - Best practises for configuration management

Henning Westerholt

Kamailio World

September 2022 - Online

Agenda

- About
- Background
- General remarks
- Building blocks
- Examples
- Contact

About GILAWA

- We offer services for Real-Time Communication platforms
  - Consulting and Management
  - Administration/Developer trainings
  - Development and IT Operations
- Kamailio experience since 2007
- Independent and neutral service provider
  - No own end-user products
  - No vendor contracts
- Our customer are Internet Service Providers and Telephone Provider
- Mainly in Germany, Europe and North-America

General remarks

- Best practices can be subjective, presentation about what worked well for us
- The examples will use the standard Kamailio configuration language and ansible as configuration management framework
- Usually this can be adapted easily to other languages or frameworks
- The presentation will show several examples to describe the different ideas
- The examples are more targeted to traditional deployments, but in the last example an approach for docker/Kubernetes environments will be shown

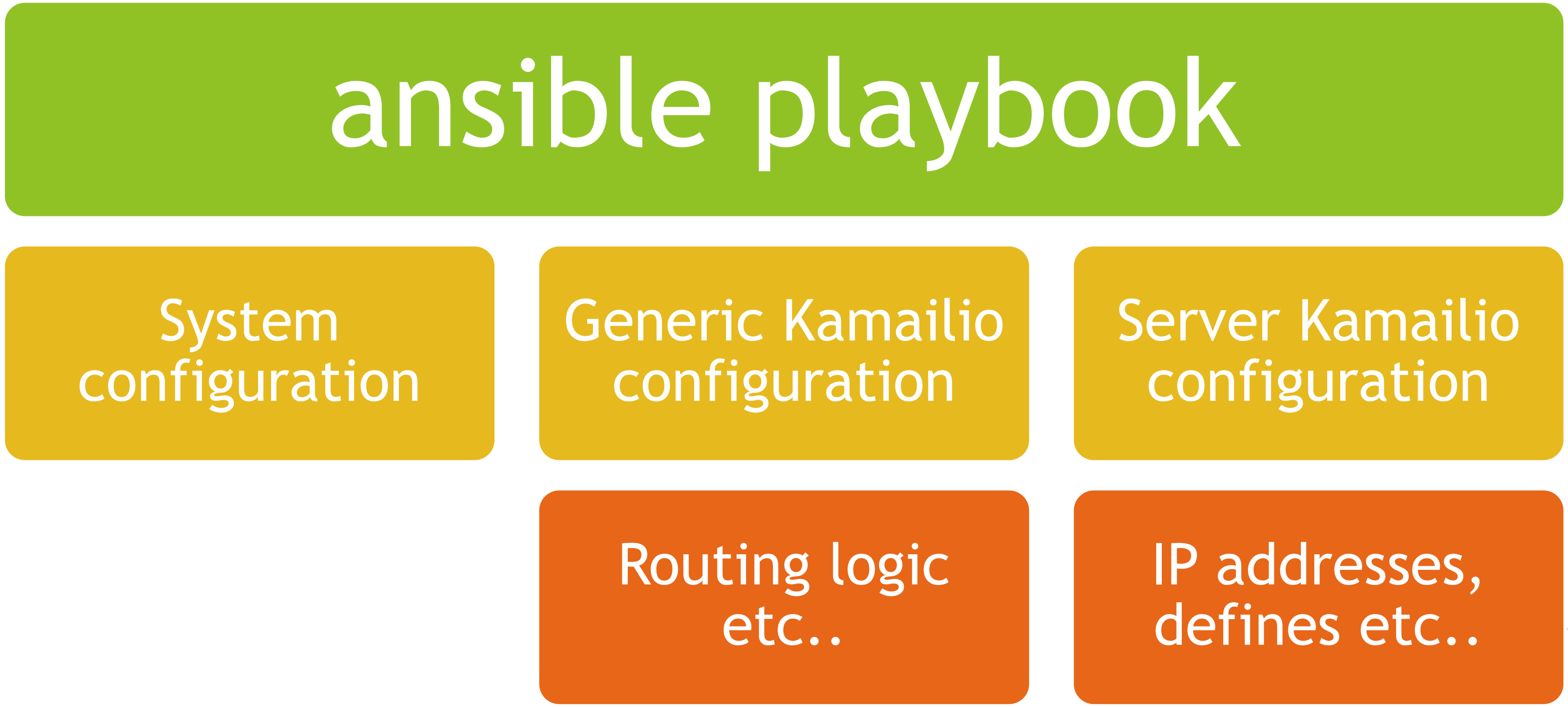
Kamailio and ansible building blocks

- Kamailio offers a wide range of building blocks to modularize your configuration
- They work similar to other pre-processing tools e.g. in C/C++
- Examples:
  - #!ifdef and #!else
  - #!define and/or #!subst
  - #include_file and #import_file
- This overlaps of course with template languages of configuration management frameworks like ansible, which uses jinja2
- Suggestion to keep to only one template language per file if possible
- For ansible variables use the standard approach with host or group variables and vaults for secret data

Example 1 – Common deployment (1/3)

- Most people starts with a small deployment, like a few Kamailio servers
- Common scenarios are load-balancer, registration or call-routing services
- Even for small scenarios like this you should use a configuration management framework to deploy your services
  - Standard installations that are repeatable
  - Configuration management as documentation
  - Foundation for later extensions or scaling
- Keep the configuration and configuration management files in git
- In ansible you usually start with a small playbook that does everything
- Keep it simple in the beginning

Example 1 – Common deployment (2/3)


ansible playbook
System configuration
Generic Kamailio configuration
Server Kamailio configuration
Routing logic etc..
IP addresses, defines etc..

Example 1 – Common deployment (3/3)

- Generic Kamailio configuration
  - It should only contain your routing logic in the individual routes
  - If you start from scratch, the default configuration is usually a good start
  - Make sure to have proper whitespace formatting, e.g. code blocks indention
  - It should not contain server specific information or logic specific to the server
- Server Kamailio configuration
  - Contains information specific to the server, like IP addresses, networks etc..
  - Some parts can be specified in the configuration, some parts needs a #!define
  - Use #!ifdef to activate or deactivate certain functionality in the generic configuration

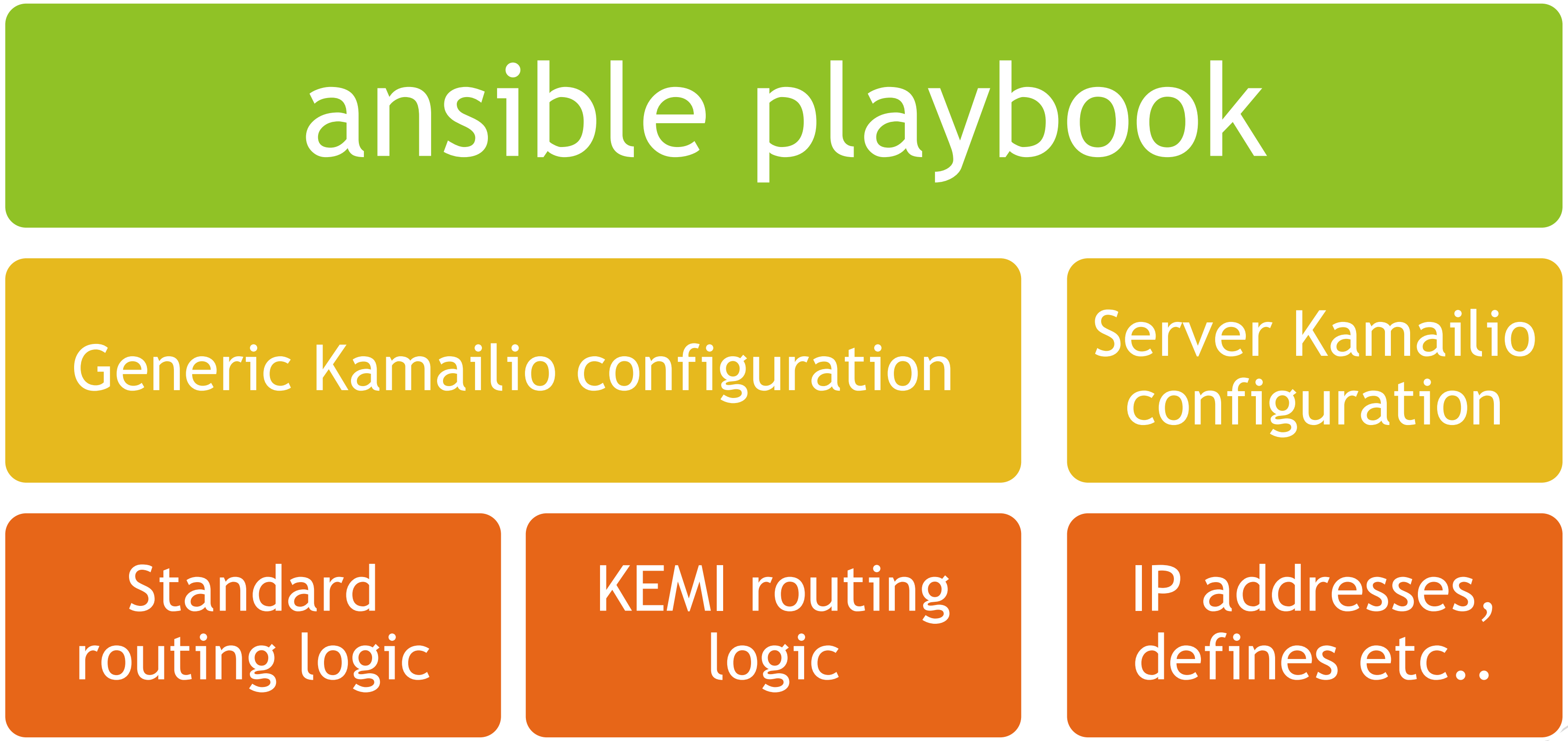
Example common server configuration

```
debug={{ kamailio_log_level }}
{% if presence == true %}
#!define WITH_PRESENCE
{% endif %}
{% if monitoring == true %}
#!define WITH_JSONRPC
{% endif %}
{% if advertise_ip is defined %}
listen=udp:{{ private_ip }}:5060 advertise {{ advertise_ip }}:5060
{% else %}
listen=udp:{{ public_ip }}:5060
{% endif %}
#!subst "!DBURL!{{ mysql_dburl }}!g"
```

Example 2 – Mix between standard and KEMI style configuration

- If you work with developer coming from a non-telco background, they will usually prefer the modern KEMI style configuration (e.g. lua, python etc..)
- If you have an existing configuration, they are written usually in the standard configuration language
- You can extend your existing configuration with the KEMI configuration parts
- Choose and standardize on one KEMI language, do not mix different languages
- Try to unify and consolidate the existing configuration to the KEMI approach in the medium to long term, as it increases complexity considerably
- If you need only one particular function, consider using the standard language, it can be pretty powerful
- Try to keep one direction, e.g. standard language calls sub-route in KEMI

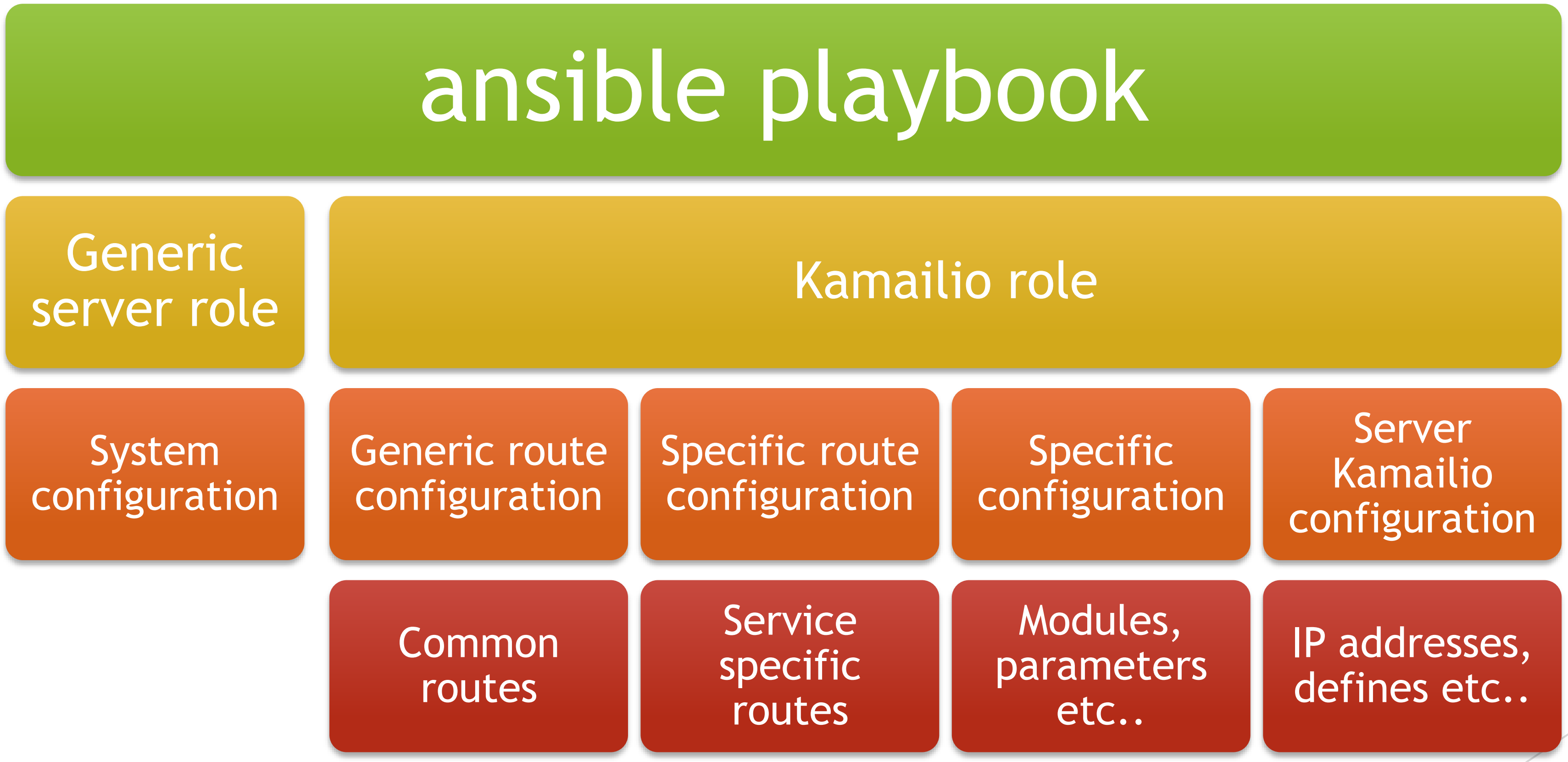
Example 2 – Mixed language deployment


ansible playbook
Generic Kamailio configuration
Server Kamailio configuration
Standard routing logic
KEMI routing logic
IP addresses, defines etc..

Example 3 – Large scale deployment with multiple regions and/or functions

- If your customer base grows, you usually also need to scale your service
- Many people just copy and paste the existing configuration and adapt it
- This approach of course will lead to large code duplication and problems
- Doing a later refactoring is a lot of work, so in best case you start with modularization of the configuration before
- Usually people organize according the service type or location
- Also keep the configuration management in mind, usage of ansible roles are a good option
- You can pass data between the different routes by (X)AVPs, variables etc..

Example 3 – large scale deployment


ansible playbook
Generic server role
Kamailio role
System configuration
Generic route configuration
Specific route configuration
Specific configuration
Server Kamailio configuration
Common routes
Service specific routes
Modules, parameters etc..
IP addresses, defines etc..

Example larger server configuration

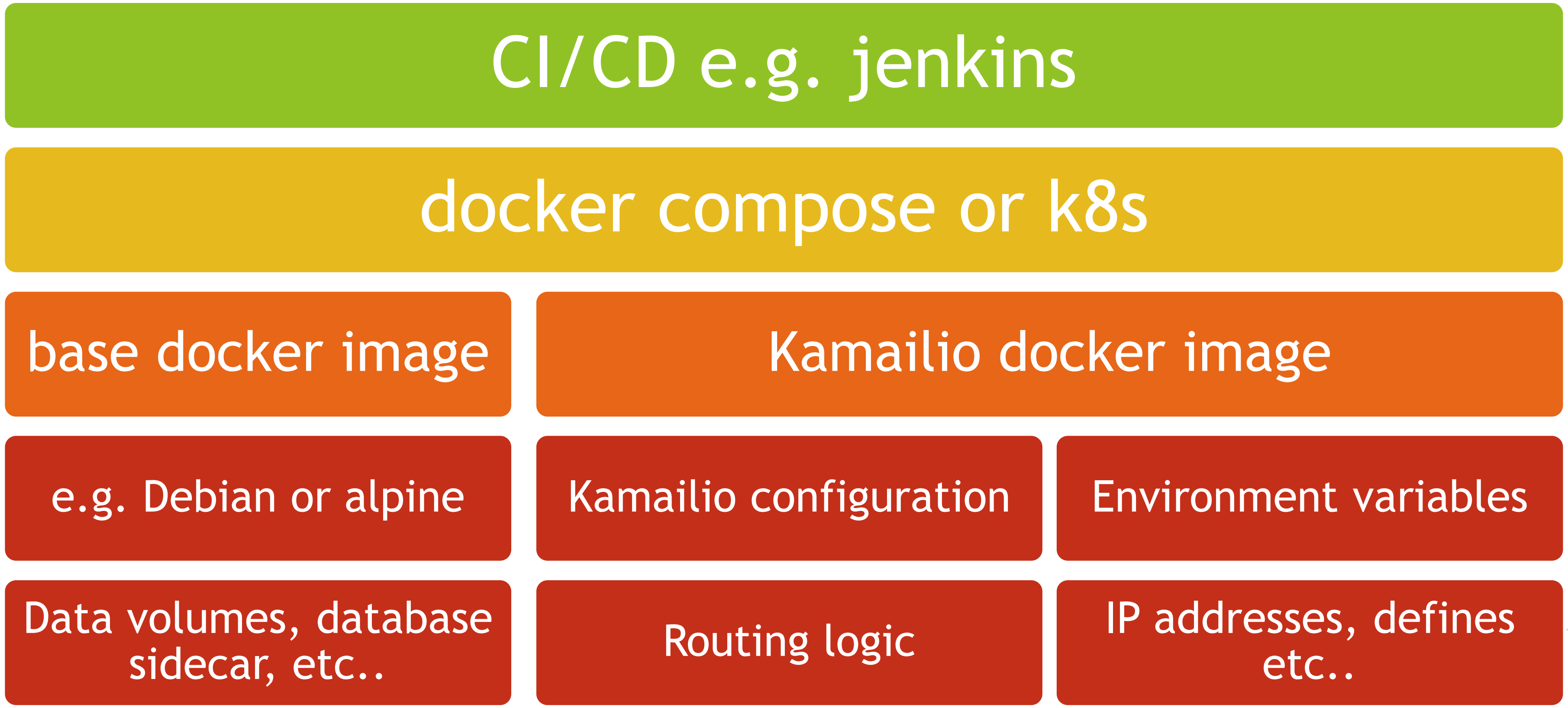
```
include_file "server.cfg-{{ environment }}-{{ service }}“
include_file "common-{{ service }}.cfg“
include_file "route.cfg“
include_file "route-{{ service }}.cfg“

request_route {
  route(main_route);
  exit;
}
```

Example 4 – Deployment with docker or Kubernetes

- If you use docker for deployment, you usually end up doing your own docker packages as you need a specific configuration for your service
- Kamailio supports now many pre-processing as command line parameter
  - Setting a define with “-A”
  - Setting a subst with “--subst”
- Recent Kamailio version also support “modparamx” for dynamic variable parameter evaluation
- With Kubernetes you usually will end up using environment variables for this information
- You could of course use a ansible to build the docker commands, but usually a CI/CD infrastructure is used
- So preferable try to not mix the different approaches

Example 4 – docker or k8s deployment



Thank you

- Thank you – any questions?
- What is your experience with configuration management and Kamailio?
- Any other opinion or remarks on certain topics?

GILAWA

Thank you

Henning Westerholt

GILAWA Ltd

hw@gilawa.com

https://gilawa.com/