Database Ops (DbOps)

Kremenjaš, Davorin

Conference presentation / Izlaganje na skupu

Database Ops (DbOps)

Davorin Kremenjaš
University Computing Centre (Srce)
Zagreb (Croatia)

Informix®

International Informix Users Group
www.iiug.org

Agenda

- Status
- Background
- Motivation
- Tools
- Implementation
- Observability
- Ecosystem
- Future plans
- Q&A

Status

- Terminology/definition overload
  - Database DevOps
  - Database Ops (DbOps)
  - Data Ops (only for analytics?)
  - A subset of SysOps?
  - Can it fit into GitOps?
  - Is it just about automation?
- Main objectives
  - apply DevOps principles and tools in day-to-day DB admin/dev tasks
  - delegate (some of the) DBA workload to developers but...
  - relieve developers of need to know the DB internals
  - while maintaining security aspects
- Bonus features
  - Integrate databases into existing automated development pipelines
  - expose data sources as first class building blocks
  - streamline release management

Background

- company: ~150 employees, 50 years
  - national IT infrastructure provider for science & higher education
- team: ~30 employees, bespoke information systems (on top of infrastructure)
- a number of existing, large, bespoke systems written in Java
  - desktop & web apps
- new and existing reporting/analytics systems
  - MS SQL data warehouse
  - R/Shiny standalone apps
- very diverse team
  - infrastructure/virtualization team
  - Linux sysadmins
  - backup/archive admins
  - Java desktop devs
  - Java web devs
  - R devs/analysts
  - and a single DbOps/DevOps guy…
- must have: common tools and interfaces for all teams
- cloud-native yet to come
  - still very much VM users
  - only a handful of docker-ized, manually maintained services deployed to production

Motivation

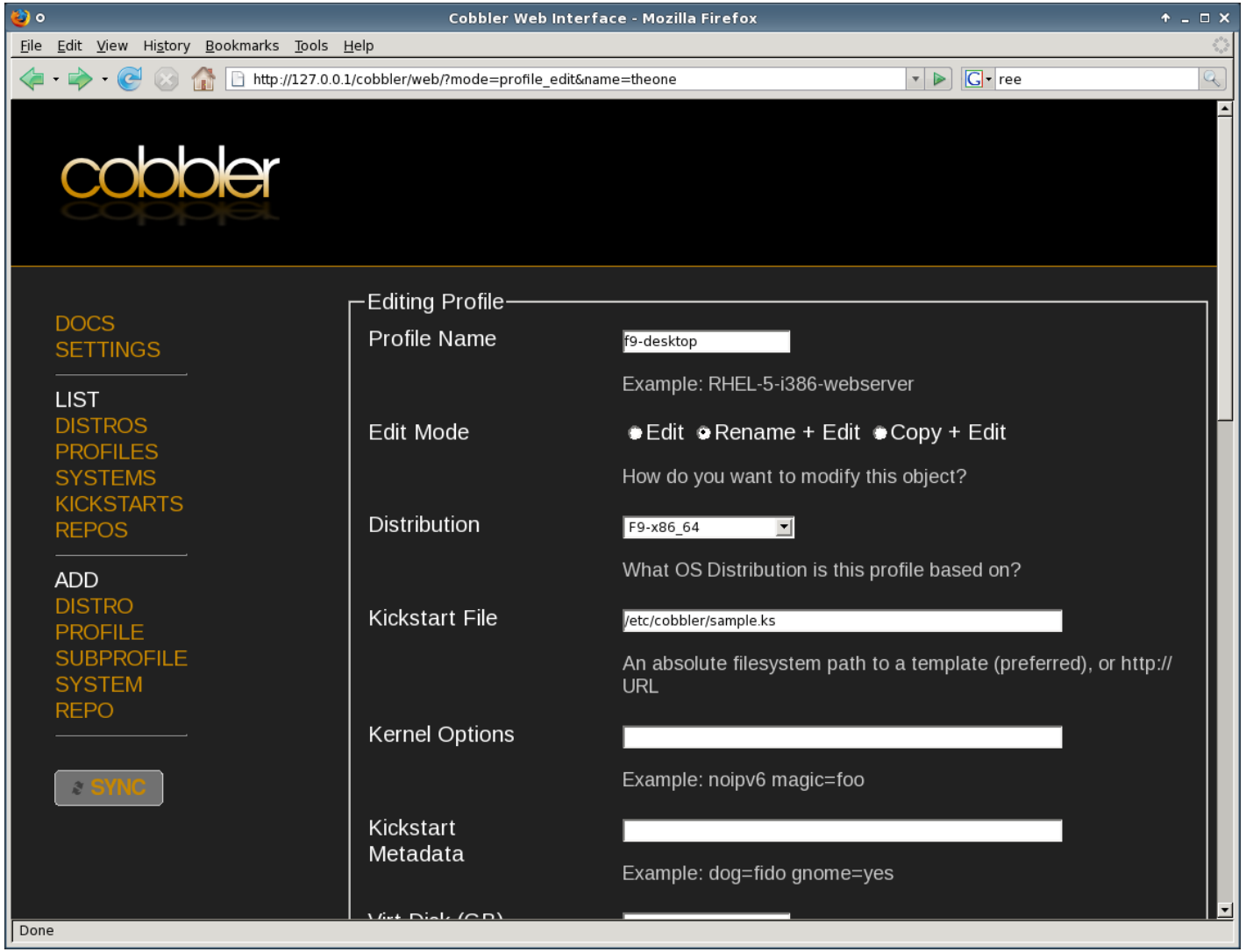
- reuse existing DBA expertise
- reuse existing scripts
  - utils2_ak – most generic functionality already there
  - in-house built – for remaining bespoke needs
    - mainly shell via cron
- expose through common, simple to use interface
- enforce security through standard/existing mechanisms
    - *nix permissions
    - sudoers rules
- visualize through standard tools
- break-up knowledge silos
- develop new organizational workflows
  - horizontal
  - visible
  - auditable
  - secure

Tools

- get Informix VM up and running
  - VM commissioning: Vmware virtualization platform
  - VM install: Cobbler (https://cobbler.github.io/)
  - VM configure: Puppet (https://puppet.com/open-source/)
- get Informix instance up and running
  - Informix configure: in-house Puppet manifests
- manage Informix instances/databases
  - Bolt: https://puppet.com/docs/bolt/latest/bolt.html
    - developed as a part of Puppet, but can be used as a standalone tool
  - instance level ops
    - stop/start/restart/status/log
  - db level ops
    - copy (source db from one instance to dest db on another instance)
    - build (new empty db from Git repo)
    - support ETL steps
    - run ad-hoc query
    - run SQL script(s)
- coordinate & visualize low-level ops above
  - Gitlab CI (https://docs.gitlab.com/ee/ci/)

DB VM install (Cobbler)

- Linux installation server
- rapid setup
  - rolling out new systems
  - changing existing ones
- glues together and automates:
  - installation
  - DNS
  - DHCP
  - package updates
  - power management
  - configuration management
  - orchestration
  - …
- old-ish web UI – but it's not about a good looking GUI anyway



DB VM install (Cobbler)

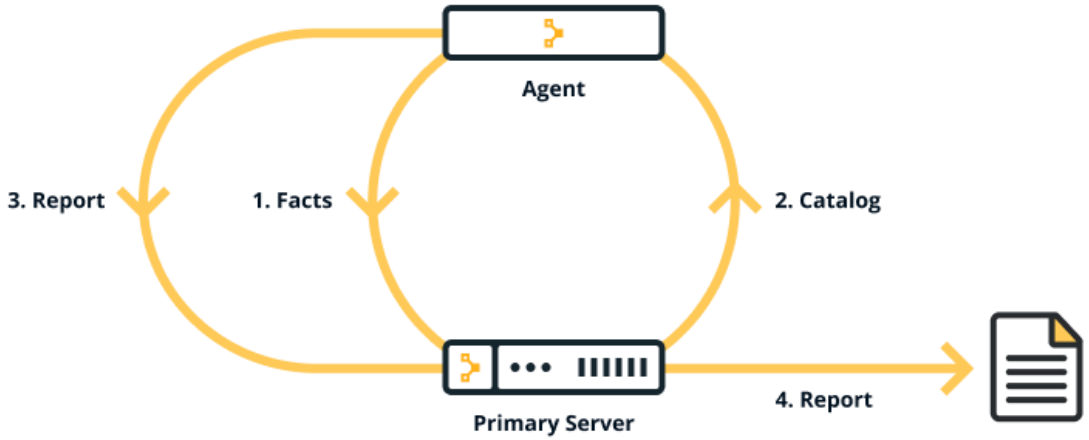
```
cobblersystem { 'db-dev-1.croris':
    ensure     => present,
    profile    => 'CentOS-7-x86_64-srce',
    interfaces => { 'eth0'   =>
                    { mac_address => '00:50:56:ad:ea:0d',
                      ip_address => '10.25.7.140',
                      netmask => '255.255.248.0',
                      static => true, management => true,
                    },
                  },
    gateway    => '10.25.0.1',
    hostname   => 'db-dev-1.croris',
    require    => Service[$::cobbler::service_name],
  }

  cobblerprofile { 'CentOS-7-x86_64-srce':
    ensure      => present,
    distro      => 'CentOS-7.7-x86_64',
    nameservers => $::cobbler::nameservers,
    repos       => 'PuppetLabs-PC1-7-x86_64',
    kickstart   => '/var/lib/cobbler/kickstarts/CentOS-7-x86_64-srce.ks',
    kernel_options =>  { 'net.ifnames' =>  '0', biosdevname   =>  '0',
    },
  }

cobbler::add_distro { 'CentOS-7.7-x86_64':
    arch              => 'x86_64',
    isolink           => 'http://mirror.centos.plus.hr/centos/7.7.1908/isos/x86_64/CentOS-7-x86_64-DVD-1908.iso',
    include_kickstart => false,
  }
```

Informix configuration (Puppet)

- manage and automate server configuration
- desired state configuration
- server/agent architecture
- declarative DSL
- consistency & automation
- idempotence and versioning
- simple and extensible



Informix configuration (Puppet)

- initial IDS/GLS binaries install still manual
- custom written Puppet manifests
- one input parameter only – instance name – the rest is all automatic:
  - PAM config (LDAP/RADIUS user authentication)
  - OS package dependencies (ksh, libaio, ...)
  - users:
    - informix – DBSA
    - horvat – DBA
  - group (dba)
  - directory structure
    - /etc/informix
    - /var/log/informix
    - chunk dir (/data/ifxinstance)
  - config/environment files
    - /etc/profile.d/informix.sh – (almost) all Informix env. variables (except locales)
    - /etc/informix/sqlhosts
    - systemd unit files (IDS and HQ)
  - backup/archive config
    - TSM policy and environment setup

Informix configuration (Puppet)

```
class informix {

  $instance = hiera('informix::instance')

  ## users and groups
  include user::informix
  include user::horvat
  include group::dba

  ## directories
  file { 'etc_informix':
    ensure => directory,
    path   => '/etc/informix',
    owner  => root,
    group  => root,
    mode   => '0755',
  }
  ...

  ## configuration
  file { '/etc/profile.d/informix.sh':
    ensure  => present,
    owner   => informix,
    group   => dba,
    mode    => '0640',
    content => template('informix/informix.sh.erb'),
  }

  ...
```

Informix configuration (Puppet)

```
class informix {

  ## service
  systemd::unit_file { 'informix.service':
    source  => 'puppet:///modules/informix/informix.service',
    require => File['/etc/profile.d/informix.sh'],
  }
  service { 'informix.service':
    ensure  => running,
    enable  => true,
    require => [Systemd::Unit_file['informix.service'], File['/etc/profile.d/informix.sh']],
  }
  ## packages
  if $facts['os']['family'] == 'Redhat' and $facts['os']['release']['major'] == '7' {
    package { 'libaio': ensure => present, }
    package { 'ksh': ensure => present, }
  }
}

template('informix/informix.sh.erb')

INFORMIXSERVER=<%= @instance %>
INFORMIXDIR=/usr/informix
INFORMIXSQLHOSTS=/etc/informix/sqlhosts
ONCONFIG=onconfig.<%= @instance %>
DBEDIT=vi
NODEFDAC=yes

export INFORMIXSERVER INFORMIXDIR INFORMIXSQLHOSTS ONCONFIG DBEDIT NODEFDAC
```

Informix instances/databases Management (Bolt)

- agent-less
- inventory based
  - user defined host/target inventory and grouping
- minimal setup
- enables existing script reuse (shell, perl, python, ruby, …)
  - only limited by what target nodes can run
- a hierarchy of “action” objects building on each other
  - commands
  - tasks
  - plans
- out-of-the box integration with Puppet

Informix instances/databases Management (Bolt)

- Bolt: https://puppet.com/docs/bolt/latest/bolt.html
- instance level ops
  - stop/start/restart/status/log
- db level ops
  - copy (source db from one instance to dest db on another instance)
  - build (new empty db from Git repo)
  - support ETL steps
  - run ad-hoc query
  - run SQL script(s)

Manage Informix instances/databases (Bolt)

```
dkremenjas@dk-tc-m910t:~/Boltdir$ bolt inventory show

Targets
  admindev
  portaldev
  admintest
  portaltest
  adminrun
  portalrun
  crorisdev
  croristest
  crorisrun
  analyticsdev
  analyticsrun
  dockerdev
  dockerrun

dkremenjas@dk-tc-m910t:~/Boltdir$ bolt group show

Groups
  all
    app
      appdev
      apptest
      apprun
    db
      dbdev
      dbtest
      dbrun
    docker
```

Manage Informix instances/databases (Bolt)

```
dkremenjas@dk-tc-m910t:~/Boltdir$ bolt command run 'uptime' --targets appdev
Finished on portaldev:
   13:17:02 up 22 days,  6:47,  0 users,  load average: 0,01, 0,10, 0,08
Finished on admindev:
   13:17:02 up 69 days,  6:36,  0 users,  load average: 0,00, 0,01, 0,00
Successful on 2 targets: admindev,portaldev


dkremenjas@dk-tc-m910t:~/Boltdir$ bolt task show
Tasks
  ...
  db::build_and_run_sql           DB build and run ad-hoc SQL
  db::create                      DB Create
  db::drop                        DB Drop
  db::log                         DB Log
  db::rename                      DB Rename
  db::run                         DB run ad-hoc SQL
  db::run_script                  DB run SQL script
  db::start                       DB Start
  db::status                      DB Status
  ...

dkremenjas@dk-tc-m910t:~/Boltdir$ bolt plan show
Plans
  db::build                       Build a DB from source
  db::copy                        Run a DB copy (in dirty read mode)
  db::etl                         Run ETL step
  db::run_scripts                 Run a set of SQL scripts on a specified target/database
  docker::reload_stack            Reload a Docker Stack
  docker::restart_services        Restart a set of Docker services on a specified target/stack
```

Manage Informix instances/databases (Bolt)

```
dkremenjas@dk-tc-m910t:~/Boltdir$ bolt task show db::run_script
db::run_script
  DB run SQL script

Usage
  bolt task run db::run_script --targets <targets> dbname=<value> script=<value> failfast=<value>

Parameters
  dbname  String[1]
    Database Name

  script  String[1]
    SQL script

  failfast  Boolean
    Fail multi-SQL scripts on first error
    Default: false
```

#run_script.sh

```
#!/usr/bin/env bash
if ${PT_failfast}
then
  output=$( sudo -inku horvat DBACCNOIGN=1 dbaccess ${PT_dbname} ${PT_script} 2>&1 )
else
  output=$( sudo -inku horvat dbaccess ${PT_dbname} ${PT_script} 2>&1 )
fi
rc=$?
/bin/echo ${output}
exit ${rc}
```

Manage Informix instances/databases (Bolt)

```
dkremenjas@dk-tc-m910t:~/Boltdir$ bolt task show db::log
db::log
  Show last lines of DB log file

Usage
  bolt task run db::log --targets <targets>

# log.sh
#!/usr/bin/env bash
sudo -inku informix onstat -m
```

```
dkremenjas@dk-tc-m910t:~/Boltdir$ bolt task show db::restart
db::restart
  DB Restart

Usage
  bolt task run db::restart --targets <targets>

# restart.sh
#!/usr/bin/env bash
sudo systemctl restart informix.service
```

Manage Informix instances/databases (Bolt)

```
dkremenjas@dk-tc-m910t:~/Boltdir$ bolt plan show
Plans
  db::build                          Build a DB from source
  db::copy                           Run a DB copy (in dirty read mode)
  db::etl                            Run ETL step
  db::run_scripts                    Run a set of SQL scripts
  docker::reload_stack               Reload a Docker Stack
  docker::restart_services           Restart a set of Docker services

dkremenjas@dk-tc-m910t:~/Boltdir$ bolt plan show db::etl
db::etl
  Run ETL step

Usage
  bolt plan run db::etl targets=<value> dbname=<value> src=<value> [failfast=<value>]

Parameters
  targets  TargetSpec
    Target DB Server

  dbname  String
    Target DB Name

  src  String
    Absolute path to the file containing the ETL SQL script

  failfast  Boolean
    Fail multi-SQL scripts on first error
    Default: false
```

Manage Informix instances/databases (Bolt)

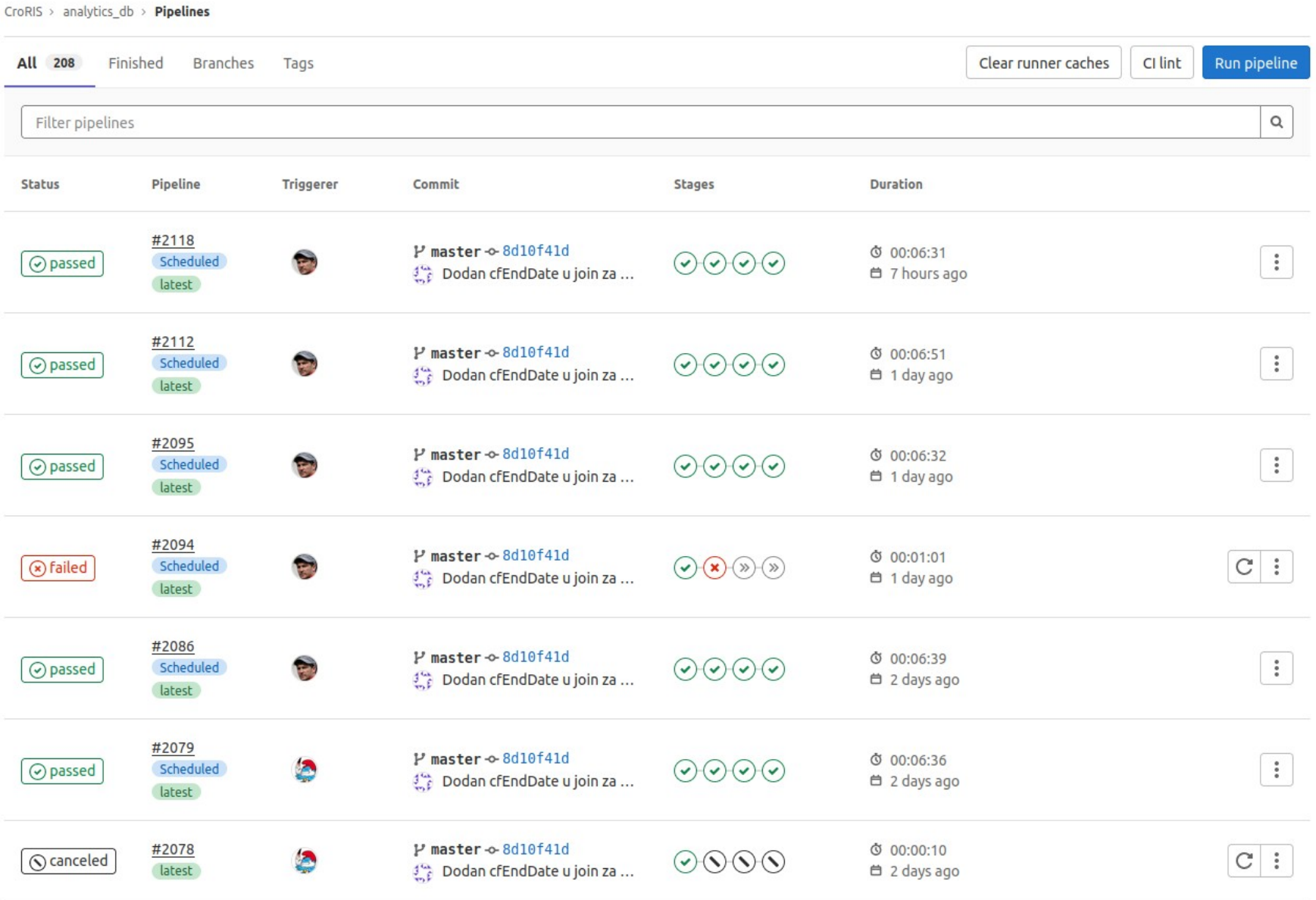
db::etl

```
description: Run ETL step
parameters:
  targets:
    type: TargetSpec
    description: "Target DB Server"
  dbname:
    type: String
    description: "Target DB Name"
  src:
    type: String
    description: "Absolute path to the file containing the ETL SQL script"
  failfast:
    type: Boolean
    description: "Fail multi-SQL scripts on first error"
    default: false
```

Manage Informix instances/databases (Bolt)

```
steps:
  - message: Running ETL from $src script...
  - command: /usr/bin/echo -n `/usr/bin/date +%Y%m%d%H%M%S%N`
    description: "Get time down to nanoseconds to use as a seed in the next step"
    targets: $targets
    name: seed
  - eval: $seed[0]['stdout']
    name: suffix
    description: "Extract the result contained in the standard output of the previous step"
  - upload: $src
    destination: "/tmp/script${suffix}.sql"
    targets: $targets
    description: "Upload SQL script file using upload task"
  - task: db::run_script
    name: db_run_script
    description: "Run uploaded DB script using db::run_script task"
    targets: $targets
    parameters:
      dbname: $dbname
      script: "/tmp/script${suffix}.sql"
      failfast: $failfast
  - command: "/usr/bin/rm /tmp/script${suffix}.sql"
    targets: $targets
    description: "Clean up"
```

Coordinate & Visualize (Gitlab CI)



Coordinate & Visualize (Gitlab CI)

```
image: "docker-registry.srce.hr/croris/croris_parent/croris"

include:
  - project: croris/utilities
    file:
      - gitlab/ci-templates/.setup_ssh.yml

variables:
  USER: ci-croris

stages:
  - setup
  - build
  - load

# set up env vars to be used in subsequent stages
setup:
  extends: .setup_ssh
  stage: setup
  script:
    - echo "DB_LOCALE=hr_HR.utf8" > setup.env
    - echo "CLIENT_LOCALE=hr_HR.utf8" >> setup.env
    - echo "TIMESTAMP=$(/bin/date +%m%d%H%M%S)" >> setup.env
    - git clone git@gitlab.srce.hr:croris/utilities.git
  artifacts:
    paths:
      - utilities/
    reports:
      dotenv: setup.env
```

Coordinate & Visualize (Gitlab CI)

```
# create a mirror DB
.do_mirror:
  script:
    # symlink Bolt embedded project directory
    - ln -sf utilities/bolt Boltdir
    - cd Boltdir
    # rename old mirror DB
    - USER=${USER} bolt task run db::rename --targets ${TARGET} curdbname=${MIRRORDB} newdbname=${MIRRORDB}${TIMESTAMP}
    # create new mirror DB as a copy
    - USER=${USER} bolt plan run db::copy source=${SOURCE} srcdbname=${SRCDB} target=${TARGET} tgtdbname=${MIRRORDB}

# build the empty analytics DB
.do_build:
  after_script:
    # symlink Bolt embedded project directory
    - ln -sf utilities/bolt Boltdir
    - USER=${USER} bolt task run db::drop   --targets ${TARGET} dbname=${DBNAME}
    - USER=${USER} bolt task run db::create --targets ${TARGET} dbname=${DBNAME}
    - USER=${USER} bolt plan run db::etl    --targets ${TARGET} dbname=${DBNAME} src="${CI_PROJECT_DIR}/a_DbStruct.sql"
    - USER=${USER} bolt plan run db::etl    --targets ${TARGET} dbname=${DBNAME} src="${CI_PROJECT_DIR}/b_DbLogs.sql"
    - USER=${USER} bolt plan run db::etl    --targets ${TARGET} dbname=${DBNAME} src="${CI_PROJECT_DIR}/c_DbProcedure.sql"
    - USER=${USER} bolt task run db::run    --targets ${TARGET} dbname=${DBNAME}
                                                                sql="EXECUTE PROCEDURE createSynonyms('${MIRRORDB}')"
    - USER=${USER} bolt plan run db::etl    --targets ${TARGET} dbname=${DBNAME} src="${CI_PROJECT_DIR}/d_DbPermissions.sql"
```

Coordinate & Visualize (Gitlab CI)

```
# load the analytics DB
.do_load:
  script:
    # symlink Bolt embedded project directory
    - ln -sf utilities/bolt Boltdir
    - USER=${USER} bolt plan run db::etl --targets ${TARGET} dbname=${DBNAME}
                                                             src="${CI_PROJECT_DIR}/etl/01_LoadDimDatum.sql"
    - USER=${USER} bolt plan run db::etl --targets ${TARGET} dbname=${DBNAME}
                                                             src="${CI_PROJECT_DIR}/etl/02_LoadDimTables.sql"
    - USER=${USER} bolt plan run db::etl --targets ${TARGET} dbname=${DBNAME}
                                                             src="${CI_PROJECT_DIR}/etl/04_LoadFactTables.sql"
    - USER=${USER} bolt plan run db::etl --targets ${TARGET} dbname=${DBNAME}
                                                             src="${CI_PROJECT_DIR}/etl/05_LoadBridgeTables.sql"
```

Coordinate & Visualize (Gitlab CI)

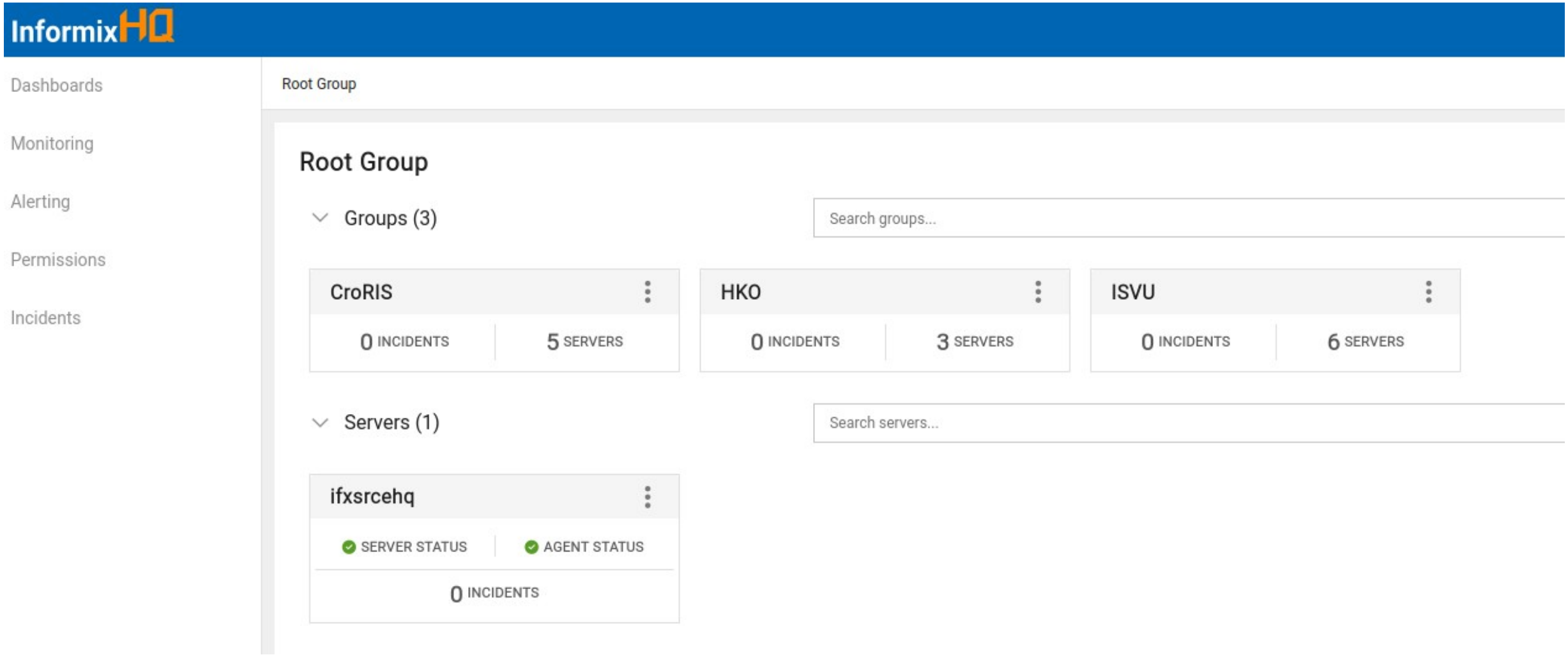
```
build_dev:
  extends:
    - .setup_ssh
    - .do_mirror
    - .do_build
  stage: build
  variables:
    DBNAME: reportsdev
    SOURCE: dbdev
    TARGET: analyticsdev
    SRCDB: dbdev
    MIRRORDB: dbdevmirror
  environment:
    name: dev
  except:
    - schedules

load_dev:
  extends:
    - .setup_ssh
    - .do_load
  stage: load
  variables:
    TARGET: analyticsdev
    DBNAME: reportsdev
  environment:
    name: dev
  except:
    - schedules
```

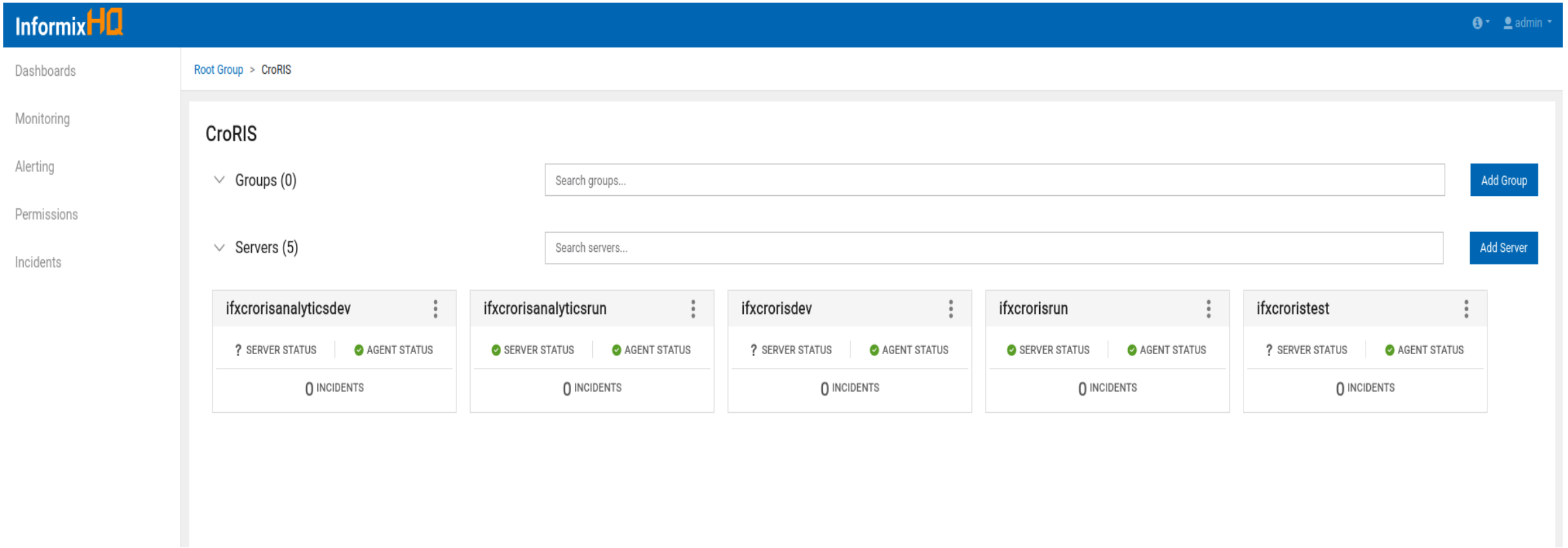
```
build_run:
  extends:
    - .setup_ssh
    - .do_mirror
    - .do_build
  stage: build
  variables:
    DBNAME: reports
    SOURCE: dbrun
    TARGET: analyticsrun
    SRCDB: db
    MIRRORDB: dbmirror
  environment:
    name: run
  only:
    - schedules

load_run:
  extends:
    - .setup_ssh
    - .do_load
  stage: load
  variables:
    TARGET: analyticsrun
    DBNAME: reports
  environment:
    name: run
  only:
    - schedules
```

Observability (Informix HQ)



Observability (Informix HQ)



Observability (Informix HQ)

- planned to use Nagios & Zabbix (long established company-wide monitoring tools)
  - poor out-of-the-box Informix support
  - a lot of manual work needed
- handled by Informix HQ instead
- agent - automatically configured and started by Puppet via systemd services
- server - manual addition for each configured agent

```
class informix {
...
## service informix-hq-agent
  systemd::unit_file { 'informix-hq-agent.service':
    source  => 'puppet:///modules/informix/informix-hq-agent.service',
    require => File['/etc/profile.d/informix.sh'],
  }

  service { 'informix-hq-agent.service':
    ensure  => running,
    enable  => true,
    require => Systemd::Unit_file['informix-hq-agent.service'],
  }
...
}
```

Ecosystem

- web layer (Apache/HAProxy)
- apps (Java/R)
- Docker
- → all handled via Puppet configs
- → most frequently used actions handled by Bolt

```
dkremenjas@dk-tc-m910t:~/Boltdir$ bolt <task|plan> show

Tasks
  app::log                        App Log
  app::start                      Start App
  app::start_all                  Start ALL Apps
  app::status                     App Status
  app::status_all                 Status ALL Apps
  app::stop                       Stop App
  app::stop_all                   Stop ALL Apps
...
  docker::restart_service         Docker: Service Restart
...
  web::reload                     Web Server Reload

Plans
...
  docker::reload_stack           Reload a Docker Stack
  docker::restart_services       Restart a set of Docker services
```

Ecosystem

app::status

```
#!/usr/bin/env bash
json=$(</dev/stdin)
appname=$(/bin/echo $json | /usr/bin/jq '.appname' | /usr/bin/tr -d \")
profile=$(/bin/echo $json | /usr/bin/jq '.profile' | /usr/bin/tr -d \")
minutes=$(/bin/echo $json | /usr/bin/jq '.minutes' | /usr/bin/tr -d \")
sudo journalctl -u ${profile}@${appname}.service --since="${minutes}minutes ago" -l --no-pager
```

docker::restart_service

```
#!/usr/bin/env bash
sudo /usr/bin/docker service update --force --update-parallelism 1 --update-delay 0s ${PT_service}
```

web::reload

```
#!/usr/bin/env bash
sudo systemctl reload apache2.service
```

Future Plans

- support Informix silent install
  - ids_install extract package (aka “-DLEGACY”)
  - store within Cobbler repo (similar to OS ISO images)
  - run install as a part of the VM creation/configuration process
- onconfig templating
- Informix HQ admin/monitor user(s) permission setup
  - account creation (mapped user)
  - access to sysuser/sysmaster/sysadmin DBs
- release/schema management
  - currently manual/crude
    - accumulate production schema/data changes in SQL script
    - take downtime (if needed) and apply
    - flush the SQL script and start over
  - looking at automation tools
    - Liquibase
    - Flyway
- Edition specific schema transformations
  - e.g. copying tables from EE to IE → must remove fragmentation
- Puppet Enterprise?
  - Web GUI for everything

Conclusion

- 90%+ of DB install/config/maintain/use tasks automated
  - minutes instead of hours/days
- complex workflows visualized
- “everyone is a DBA”
- 100% open source tooling
  - used and trusted by big names
  - reused established company-wide tools rather than inventing our own
  - simple to extend (everyone reads/writes simple YAML/shell)
- unified developer experience
  - working across diverse set of technologies
- DB code treated on par with app code
  - each commit/push/merge to DB code repositories main branch tested
  - tight feedback loop
- quick newcomer on-boarding
- breaking down knowledge silos
- sense of ownership

Questions?