

Foundry

1 **Wat is Foundry?**

2 **Demo: node create**

3 **Hoe werkt Foundry?**

4 **Waarom Laravel?**

5 **Demo: retries**

1

What is Foundry?

Foundry: een interne infrastructuur tool

Wat Foundry bijzonder maakt

- Eigen workflow-systeem
- 8 integraties
- Scheduled tasks
- Automatische
herbruikbaarheid met auto-
registrering tasks
- Dropdowns met datasets

2

Demo: node create

3

Hoe werkt Foundry?

Definiëren van actions

```
class Vm
{
  #[RequireTenant]
  #[Dataset('ip_address', [VirtualMachineDataset::class, 'cloudinitIp'])]
  public static function create(VmCreateRequest $request): CreateVm
  {
    return new CreateVm(
      data: $request->createVmPlaceholder(),
    );
  }
}
```

Reusable op 3 manieren

```
class Vm
{
  #[RequireTenant]
  #[Dataset('ip_address', [VirtualMachineDataset::class, 'cloudinitIp'])]
  public static function create(VmCreateRequest $request): CreateVm
  {
    return new CreateVm(
      data: $request->createVmPlaceholder(),
    );
  }
}
```

GUI

The screenshot displays the Foundry GUI interface for creating a bridge mapping. On the left is a navigation sidebar with a 'Bridge mappings' section expanded to show a 'Create' button. The main panel is titled 'Create bridge mapping' and contains the following fields:

- Tenant *** (dropdown menu): tenant, value: - select tenant -
- VID (vLAN ID)** (text input): vlan, value: 1-4094
- ID of the VM Cluster in Netbox *** (dropdown menu): vm_cluster, value: - choose value -
- Name of the bridge *** (text input): bridge, value: e.g. vmbr0

A 'Run command' button is located at the bottom of the form.

Reusable op 3 manieren

```
class Vm
{
  #[RequireTenant]
  #[Dataset('ip_address', [VirtualMachineDataset::class, 'cloudinitIp'])]
  public static function create(VmCreateRequest $request): CreateVm
  {
    return new CreateVm(
      data: $request->createVmPlaceholder(),
    );
  }
}
```

The screenshot shows a sidebar menu on the left with categories like Bridge mappings, Cloud Init, Devices, FHRP groups, Interscope clients, Interscope servers, IP addresses, IP prefixes, NMS servers, and Pools. The main content area is titled 'Create bridge mapping' and contains several input fields: 'Tenant' (a dropdown menu), 'VID (LAN ID)' (a text input with '1-4004' entered), 'ID of the VM Cluster in Netbox' (a dropdown menu), and 'Name of the bridge' (a text input with 'bridge' entered). A 'Run command' button is located at the bottom right of the form.

The screenshot displays the Foundry API documentation for the 'vm' endpoint. The page title is 'vm Virtual machines'. The endpoint is 'POST /vm Create virtual machine' with the path 'foundry:vm:create'. The parameters section indicates 'No parameters'. The request body is 'application/json' and is marked as 'required'. The schema section shows the following structure:

```

^ Collapse all object
tenant* ^ Collapse all string
  Tenant slug
mode* ^ Collapse all string
  Mode
  Example "cloudinit, iso"
device* ^ Collapse all integer
  ID of the Device in Netbox
root_disk_datastore ^ Collapse all string
  Name of the datastore for the root disk
memory* ^ Collapse all integer
  Memory in MB
```

API
POST /api/vm

Reusable op 3 manieren

```
class Vm
{
  #[RequireTenant]
  #[Dataset('ip_address', [VirtualMachineDataset::class, 'cloudinitIp'])]
  public static function create(VmCreateRequest $request): CreateVm
  {
    return new CreateVm(
      data: $request->createVmPlaceholder(),
    );
  }
}
```

CLI

artisan foundry:bridge-mapping:create

```
foundry-staging@admin-cap:~/foundry-staging.cyberfusion.nl$ php artisan foundry:vm:create --help
```

Description:

Create virtual machine

Usage:

```
foundry:vm:create [options] [--] <tenant>
```

Arguments:

tenant Tenant slug

Options:

```
--mode=MODE           Mode
--device=DEVICE       ID of the Device in Netbox
--root-disk-datastore=ROOT-DISK-DATASTORE  Name of the datastore for the root disk
--memory=MEMORY       Memory in MB
--hostname=HOSTNAME   Hostname
```

Gebruikersvriendelijke GUI

```
class Vm
{
  #[RequireTenant]
  #[Dataset('ip_address', [VirtualMachineDataset::class, 'cloudinitIp'])]
  public static function create(VmCreateRequest $request): CreateVm
  {
    return new CreateVm(
      data: $request->createVmPlaceholder(),
    );
  }
}
```

The screenshot shows a web form for creating a VM. The form has several fields: 'Netbox Prefix ID' (with a dropdown menu), 'DNS Name', 'Role', 'Externally managed DNS', and 'HTTP Equivalent'. The dropdown menu is open, showing a list of available prefixes with their counts and some descriptions. The first option is selected.

Netbox Prefix ID	Count / Description
2a0c:eb00:0:f6::/64	(41)
2a0c:eb00:0:f7::/64	(42, belongs to Infrastructure)
2a0c:eb00:0:f9::/64	(43)
10.40.1.0/24	(1)
10.40.4.0/24	(4)
10.40.5.0/24	(5)
10.40.6.0/24	(86)
10.40.7.0/24	(7)
10.40.8.0/24	(8)
10.40.9.0/24	(9)
10.40.10.0/24	(46)
10.40.11.0/24	(88)
10.40.12.0/24	(11)
10.40.13.0/24	(12)
10.40.14.0/24	(13)
10.40.15.0/24	(59)

Gebruikersvriendelijke GUI

```
class VirtualMachineDataset
{
  public static function cloudinitIp(?string $tenant, ?string $mode): Collection
  {
    if (! AbstractDataset::isTenantOk($tenant) || $mode !== 'cloudinit') {
      return collect();
    }

    return IpAddressDataset::search([
      'tenant' => $tenant,
      'status' => 'active',
      'family' => 4,
      'assigned_to_interface' => 'false',
    ])
    ->whereNull('assigned_object') // Filter IPs assigned to FHRP group
    ->map(IpAddressDataset::mapDisplay(...));
  }
}
```

Anatomie van een workflow

```
class CreateVm implements Workflow
{
    public function __construct(
        private readonly CreateVmPlaceholder $data,
    ) {
    }

    public static function name(): string;

    public function dependencies(): Dependencies;

    public function sequence(): Sequence;
```

Anatomie van een workflow

```
class CreateVm implements Workflow
{
    public static function name(): string
    {
        return 'Create virtual machine';
    }

    public function dependencies(): Dependencies;

    public function sequence(): Sequence;
```

Anatomie van een workflow

```
class CreateVm implements Workflow
{
    public function dependencies(): Dependencies
    {
        return new Dependencies([
            $this->tenant(...),
            $this->device(...),
            $this->vmCluster(...),
            $this->ipAddress4(...),
            $this->prefix4(...),
            $this->pool(...),
            $this->ensureDeviceHasEnoughCpus(...),
            $this->ensureDatastoreExists(...),
            $this->ensureCloudInitVendorConfigExists(...),
            $this->ensureCloudInitImageExists(...),
        ]);
    }
}
```

Anatomie van een workflow

```
class CreateVm implements Workflow
{
    public function dependencies(): Dependencies;

    private function pool(?Tenant $tenant, VmCluster $vmCluster): ?string
    {
        if ($this->data->pool === null) {
            return null;
        }

        $name = $tenant->poolName($this->data->pool);

        $existingPools = $vmCluster->proxmox()->pools()->get();
        $names = Arr::pluck($existingPools, 'poolid');

        if (! in_array($name, $names)) {
            throw DependencyValidationException::poolDoesNotExist('pool');
        }

        return $name;
    }

    public function sequence(): Sequence;
```


Anatomie van een workflow

```
class CreateVm implements Workflow
{
    public function sequence(): Sequence
    {
        return new Sequence([
            self::DETERMINE_ROOT_DISK_DATASTORE => $this->determineRootDiskDatastore(...),
            self::CLOUD_INIT_DETERMINE_NETWORKING => $this->cloudInitDetermineNetworking(...),
            self::CREATE_IN_PROXMOX => $this->createVmInProxmox(...),
            self::CREATE_IN_NETBOX => $this->createVmInNetbox(...),
            self::CLOUD_INIT_CREATE_NETWORK_INTERFACE_IN_NETBOX => $this->cloudInitCreateNetworkInterface(...),
            self::CLOUD_INIT_ASSIGN_IP_ADDRESS_V4_IN_NETBOX => $this->cloudInitAssignVmToIpAddressVersion(...),
            // and more
        ], $this->output(...));
    }

    private function output(array $carry)
    {
        return [
            'vm_id' => $carry[self::CREATE_IN_PROXMOX]->value()['vm_id'],
            'netbox_id' => $carry[self::CREATE_IN_NETBOX]->value()->id(),
        ];
    }
}
```

Anatomie van een workflow

```
class CreateVm implements Workflow
{
    public function sequence(): Sequence;

    private function cloudInitAssignVmToIpAddressVersion4(array $carry, Netbox $netbox): IpAddress
    {
        $data = $this->skipTaskUnlessCloudInit();

        /** @var VmInterface $vmInterface */
        $vmInterface = $carry[self::CLOUD_INIT_CREATE_NETWORK_INTERFACE_IN_NETBOX]->value();

        return new IpAddress(
            $netbox->ipam()->ipAddresses()->ipAddress($data->ipAddress4Id)->update([
                'assigned_object_type' => Netbox::TYPE_VM_INTERFACE,
                'assigned_object_id' => $vmInterface->id(),
            ]),
        );
    }
}
```

Scheduled tasks

```
class CheckVirtualisationResourceDistribution extends ScheduledTaskJob
{
  public static function name(): string
  {
    return CheckVirtualisationResourceDistributionAction::name();
  }

  public static function column(): string
  {
    return 'Cluster / Pool';
  }

  public static function schedule(Event $schedule): void
  {
    $schedule->daily()->onOneServer();
  }

  public static function tasks(): array
  {
    return VmCluster::where('distribution_check_enabled', true)->get()
      ->flatMap(fn (VmCluster $vmCluster) => self::handleVmCluster($vmCluster))
      ->toArray();
  }
}
```

Environment: local








Cyberfusion

- Run
- History
- Scheduled tasks
- API documentation

Check SNMP managed clients

Next: 19 June 2024 00:00:00
Cron: 0 0 * * *







Last runs

All clients	
	Missing clients       

Monitor Infscap server

Next: 19 June 2024 00:00:00
Cron: 0 0 * * *















Last runs

Server	
infscap-test.cyberfusion.nl	Failed      

Virtualisation Resource Distribution

Next: 19 June 2024 00:00:00
Cron: 0 0 * * *

Last runs

Cluster / Pool	
pve-test.cyberfusion.cloud / ACME2-bribe-insects	Requires zone migration       
pve-test.cyberfusion.cloud / ACME2-feed-badgers	No virtual machines       

4

Waarom Laravel?

Wat gebruiken we van Laravel?

- Queues, Horizon
- Service container,
- Dependency injection
- Basics: mail, events, HTTP client
- Livewire (GUI)

Advantages

- Develop faster
- Built-in logic

5

Demo: retries

Workflows

- All actions -



Search in description

Search

Name	Status	Started at	Duration	Attempts	
Foundry presenteren <i>Larafest & LarAwards 2024</i>	Success	20 June 2024 18:00:00	10 min	1	Report
Check SNMP managed clients	Missing clients	20 June 2024 00:00:08	996 ms	1	Report
Monitor Infscope server <i>infscope-test.cyberfusion.nl</i>	Success	20 June 2024 00:00:09	90 ms	1	Report