Microsoft Azure

Introducing azure-init, a minimal provisioning agent written in Rust

All Systems Go! - Berlin, Germany

- Dongsu Park
- 25. Sep. 2024

Who am I?

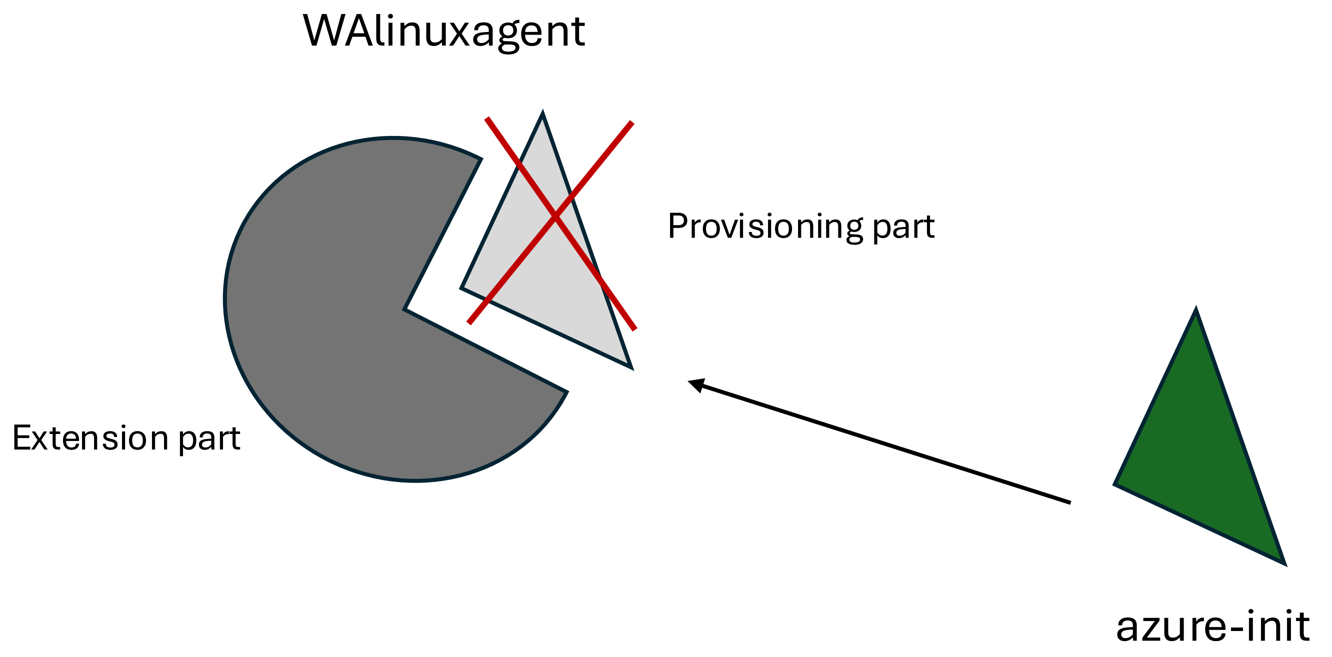
- Azure Core Linux team of Microsoft, previously Kinvolk
- Part of Flatcar Container Linux maintainers team
- Recently excited to learn Rust programming language

What is Azure-init?

- Re-implementation of provisioning agent of WALinuxAgent in Azure
  - Minimal drop-in replacement of the provisioning part
  - Written in Rust

- WALinuxAgent consists of 2 agents
  - Provisioning agent: legacy, little used by distros
    - cloud-init does the job in most cases
  - Extensions agent: used by (major) distros, fragmented, size bloat

Replace provisioning part of WAlinuxagent



Brief history of azure-init

- Started as an intern project of Cade Jacobson in Jun. 2023
  - named "*azure-provisioning-agent*"

- Maintained currently by Azure Core Linux team
  - Renamed to azure-init
  - Open-sourced with MIT license
  - Pre-alpha release 0.1.1 in Mar. 2024

Why Rust?

- Minimal binary size
  - avoid size bloat caused python runtime needed by WALinuxAgent
- Memory safety
- Growing community support
- Possible integration with other libs and SDKs in open-source ecosystem
  - e.g. Azure SDK for Rust
- Good opportunity to learn programming language

Directory tree

- libazureinit
- └── src
-     └── provision
          └── user
          └── ssh/password
          └── hostname
-      └── IMDS
-      └── wireserver
-      └── media
- config
- src
- tests
- .github

Builder-style API

- Used in libazureinit/provision
  - Not a list of parameters, but builder methods added up
  - Provisioners as builder methods

```
Provision::new(im.compute.os_profile.computer_name, user)
  .hostname_provisioners([
    #[cfg(feature = "hostnamectl")]
    HostnameProvisioner::Hostnamectl,
  ])
  .user_provisioners([
    #[cfg(feature = "useradd")]
    UserProvisioner::Useradd,
  ])
  .password_provisioners([
    #[cfg(feature = "passwd")]
  PasswordProvisioner::Passwd,
  ])
  .provision()?;
```

Unit tests

- Follows ways of native Rust unit tests

```
#[test]
fn test_pre_existing_ssh_dir() {
  let mut user =
    nix::unistd::User::from_name(whoami::username().as_str()).unwrap().unwrap();
  let home_dir = tempfile::TempDir::new().unwrap();
  user.dir = home_dir.path().into();
  std::fs::DirBuilder::new()
    .mode(0o777)
    .create(user.dir.join(".ssh").as_path()).unwrap();

  let keys = vec![
    PublicKeys {
      key_data: "not-a-real-key abc123".to_string(),
      path: "unused".to_string(),
    },
  ];
```

```
  provision_ssh(&user, &keys).unwrap();

  let ssh_dir =
    std::fs::File::open(home_dir.path().join(".ssh")).unwrap();
  assert_eq!(
    ssh_dir.metadata().unwrap().permissions(),
    Permissions::from_mode(0o040700)
  );
}
```

```
$ cargo test
```

End-to-end (Functional) tests

- Step 1: preparation of SIG (Shared Image Gallery) image
    - Create a resource account, a storage account
    - Launch a virtual machine with a given distro image
    - Create & publish Azure SIG image definition & versions

- Step 2: run actual end-to-end tests
    - "make e2e-test"
    - Launch a virtual machine
    - Build functional_test binary and copy it to the target machine
    - Run all available functional tests remotely

Demo (end-to-end test)

- recorded video

Challenges

- Minimum-Supported Rust Version
  - Not possible to simply stick with recent stable Rust version like 1.81
  - Tricky to deal with corner cases of distros stuck with older Rust
  - Support up to Rust version 12 months ago like 1.71.1
  - Add CI build and test to keep the requirements
  - Possible improvement: MSRV-aware resolver, available only in nightly Rust

Challenges

- Builder-stype API of libazureinit/provisioning
  - To be integrated with multiple distros
    - Username, group name, ssh key, hostname, etc.
  - Azure Linux (a.k.a. CBL-Mariner)
  - RHEL/CentOS
  - Debian/Ubuntu
  - Immutable OS like Flatcar, openSUSE MicroOS, etc.

Challenges

- Functional tests being CI-automated
  - Nightly CI running internally since a few weeks
  - Need to figure out how to enable on-demand e2e test while preventing abuse
  - Improvement in progress: discussion issue

Future work

- Coordinate with WALinuxAgent folks
  - Decoupling provisioning part from extensions
  - Release coordination for future release

- Add more documents
  - Getting started with development
  - Rustdocs for public functions of libazureinit API

- Provisioning telemetry via Hyper-V KVP (Key Value Pair)
  - Goal: assist with provisioning issues without requiring access to live VM
  - Open Pull Request

Questions?

Thanks!

Email: dpark@linux.microsoft.com

Mastodon: https://fosstodon.org/@dongsupark

GitHub: https://github.com/dongsupark

Contributors & community members:

Anh Vo (@anhvoms)
Bala Konda Reddy (@balakreddy)
Cade Jacobson (@cadejacobson)
Chris Patterson (@cjp256)
Jeremy Cline (@jeremycline)
Nell Shamrell-Harrington (@nellshamrell)
Peyton Robertson (@peytonr18)
Rodrigo Campos (@rata)
Sean Dougherty (@SeanDougherty)
Thilo Fromm (@t-lo)
Vincenzo Marcella (@vmarcella)

(Spare slides) Error handling

- Implemented based on thiserror crate

```
pub enum Error {
  #[error("Unable to deserialize or serialize JSON data")]
  Json(#[from] serde_json::Error),
  #[error("HTTP request did not succeed (HTTP {status} from {endpoint})")]
  HttpStatus {
    endpoint: String,
    status: reqwest::StatusCode,
  },
  #[error("executing {command} failed: {status}")]
  SubprocessFailed {
    command: String,
    status: std::process::ExitStatus,
  },
  #[error("The user {user} does not exist")]
  UserMissing { user: String },
}
```