Red Hat Developer Tools 1

Using Rust 1.52.1 Toolset

Installing and using Rust 1.52.1 Toolset
Installing and using Rust 1.52.1 Toolset

Eva-Lotte Gebhardt
egebhard@redhat.com

Olga Tikhomirova
otikhomi@redhat.com

Zuzana Zoubkova

Peter Macko

Kevin Owen

Vladimir Slavik
Abstract

Rust Toolset is a Red Hat offering for developers on the Red Hat Enterprise Linux platform. This Using Rust Toolset guide provides an overview of this product, explains how to invoke and use the different versions of Rust tools, and links to resources with more in-depth information.
# Table of Contents

**MAKING OPEN SOURCE MORE INCLUSIVE**  ................................................................. 3

**CHAPTER 1. RUST TOOLSET**  ................................................................. 4

1.1. RUST TOOLSET COMPONENTS  ......................................................... 4
1.2. COMPATIBILITY  ........................................................................ 4
1.3. GETTING ACCESS TO RUST TOOLSET ON RED HAT ENTERPRISE LINUX 7  ......................................................... 4
1.4. INSTALLING RUST TOOLSET  ......................................................... 6
1.5. INSTALLING DOCUMENTATION  ..................................................... 6
1.5.1. Installing Rust documentation  ................................................... 6
1.5.2. Installing Cargo documentation  ............................................... 7
1.6. ADDITIONAL RESOURCES  ......................................................... 7

**CHAPTER 2. THE CARGO BUILD TOOL**  ..................................................... 8

2.1. CARGO DIRECTORY STRUCTURE AND FILE PLACEMENTS  ......................... 8
2.2. CREATING A RUST PROJECT  ..................................................... 8
2.3. CREATING A LIBRARY FOR A RUST PROJECT  ...................................... 9
2.4. BUILDING A RUST PROJECT  ..................................................... 9
2.5. BUILDING A RUST PROJECT IN RELEASE MODE  ..................................... 10
2.6. RUNNING A RUST PROGRAM  .................................................. 11
2.7. TESTING A RUST PROJECT  ................................................... 11
2.8. TESTING A RUST PROJECT IN RELEASE MODE  ..................................... 12
2.9. CONFIGURING RUST PROJECT DEPENDENCIES  ..................................... 13
2.10. BUILDING DOCUMENTATION FOR A RUST PROJECT  ............................. 14
2.11. VENDORING RUST PROJECT DEPENDENCIES  ..................................... 14
2.12. ADDITIONAL RESOURCES  .................................................. 15

**CHAPTER 3. THE RUSTFMT FORMATTING TOOL**  ....................................... 16

3.1. INSTALLING RUSTFMT  ..................................................... 16
3.2. USING RUSTFMT AS A STANDALONE TOOL  ........................................ 16
3.3. USING RUSTFMT WITH CARGO  ................................................ 17
3.4. ADDITIONAL RESOURCES  .................................................. 18

**CHAPTER 4. CONTAINER IMAGES WITH RUST TOOLSET**  .......................... 19

4.1. CREATING A CONTAINER IMAGE OF RUST TOOLSET  ............................. 19
4.2. ADDITIONAL RESOURCES  .................................................. 20

**CHAPTER 5. CHANGES IN RUST 1.52.1 TOOLSET**  ....................................... 21
MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright’s message.
Rust Toolset is a Red Hat offering for developers on Red Hat Enterprise Linux. It provides the rustc compiler for the Rust programming language, the Rust package manager Cargo, the rustfmt formatting tool, and required libraries.

Rust Toolset is distributed as a part of Red Hat Developer Tools for Red Hat Enterprise Linux 7. For Red Hat Enterprise Linux 8, Rust Toolset is available as a module.

### 1.1. RUST TOOLSET COMPONENTS

The following components are available as part of Rust Toolset:

<table>
<thead>
<tr>
<th>Name</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rust</td>
<td>1.52.1</td>
<td>The Rust compiler front-end for LLVM.</td>
</tr>
<tr>
<td>cargo</td>
<td>1.52.1</td>
<td>A build system and dependency manager for Rust.</td>
</tr>
<tr>
<td>rustfmt</td>
<td>1.52.1</td>
<td>A tool for automatic formatting of Rust code.</td>
</tr>
</tbody>
</table>

### 1.2. COMPATIBILITY

Rust Toolset is available for Red Hat Enterprise Linux 7 and Red Hat Enterprise Linux 8 on the following architectures:

- AMD and Intel 64-bit architectures
- The 64-bit ARM architecture (Only RHEL 8)
- IBM Power Systems, Little Endian
- IBM Power Systems, Big Endian (Only RHEL 7)
- 64-bit IBM Z

### 1.3. GETTING ACCESS TO RUST TOOLSET ON RED HAT ENTERPRISE LINUX 7

To install Rust Toolset on Red Hat Enterprise Linux 7, you first need to access and enable Red Hat Developer Tools and Red Hat Software Collections repositories. If these repositories are already attached to your system, see Installing Rust Toolset.

**Prerequisites**

- **Wget** is installed.
  To install **Wget**, run the following command:
Procedure

1. Download the latest subscription data by running:
   ```bash
   # subscription-manager refresh
   ```

2. Register your system by running:
   ```bash
   # subscription-manager register
   ```
   As an alternative, you can also register your system by following the Registering and Unregistering a System guide.

3. Display a list of all available subscriptions and identify the pool ID by running the following command:
   ```bash
   # subscription-manager list --available
   ```
   Find the pool ID on the line beginning with Pool ID.

4. To attach the subscription that provides access to the Red Hat Developer Tools repository to your system, run the following command:
   ```bash
   # subscription-manager attach --pool=<appropriate pool ID from the subscription>
   ```
   - Replace `<appropriate pool ID from the subscription>` with the pool ID you identified in the previous step.

5. Verify which subscriptions are attached to your system by running:
   ```bash
   # sudo subscription-manager list --consumed
   ```

6. Enable the `rhel-7-variant-devtools-rpms` repository by running:
   ```bash
   # subscription-manager repos --enable rhel-7-<variant>-devtools-rpms
   ```
   - Replace `<variant>` with your Red Hat Enterprise Linux system variant (server or workstation).
     Consider using server to access the widest range of the development tools.

7. Enable the `rhel-variant-rhscl-7-rpms` repository by running:
   ```bash
   # subscription-manager repos --enable rhel-<variant>-rhscl-7-rpms
   ```
   - Replace `<variant>` with your Red Hat Enterprise Linux system variant (server or workstation).

8. Add the Red Hat Developer Tools GPG key to your system by running:
1.4. INSTALLING RUST TOOLSET

Complete the following steps to install Rust Toolset including all development and debugging tools as well as dependent packages. Note that Rust Toolset has a dependency on LLVM Toolset.

**Prerequisites**

- On Red Hat Enterprise Linux 7, a subscription providing access to the Red Hat Developer Tools content set is attached to your system. To attach the subscription, see [Getting access to Rust Toolset on Red Hat Enterprise Linux 7](#).
- All available Red Hat Enterprise Linux updates are installed.

**Procedure**

- On Red Hat Enterprise Linux 7, install the `rust-toolset-1.52` collection by running:
  
  ```bash
  # yum install rust-toolset-1.52
  ```

- On Red Hat Enterprise Linux 8, install the `rust-toolset` module by running:
  
  ```bash
  # yum module install rust-toolset
  ```

1.5. INSTALLING DOCUMENTATION

Documentation for the Rust programming language as well as Cargo is available as installable documentation. Complete the following steps to install and access the documentation on your local machine.

**Prerequisites**

- Rust is installed. For more information, see [Installing Rust Toolset](#).

**1.5.1. Installing Rust documentation**

**Procedure**

- To install the `rust-doc` package, run the following command:
  
  ```bash
  # yum install rust-toolset-1.52-rust-doc
  ```

- On Red Hat Enterprise Linux 8:
  ```bash
  # yum install rust-doc
  ```


### 1.5.2. Installing Cargo documentation

- To install the cargo-doc package, run:
  ```bash
  # yum install rust-toolset-1.52-cargo-doc
  ```

Find the *Cargo, Rust’s Package Manager* book under the following path: `/opt/rh/rust-toolset-1.52/root/usr/share/doc/cargo/html/index.html`.

- On Red Hat Enterprise Linux 8:
  ```bash
  # yum install cargo-doc
  ```

Find the *Cargo, Rust’s Package Manager* book under the following path: `/usr/share/doc/cargo/html/index.html`.

### 1.6. ADDITIONAL RESOURCES

- For more information on the Rust programming language, see the *official Rust documentation*. 
CHAPTER 2. THE CARGO BUILD TOOL

Cargo is a build tool and front end for the Rust compiler rustc as well as a package and dependency manager. It allows Rust projects to declare dependencies with specific version requirements, resolves the full dependency graph, downloads packages, and builds as well as tests your entire project.

Rust Toolset is distributed with Cargo 1.52.1.

**NOTE**

To avoid using `scl enable` in front of every command, run:

```
$ scl enable rust-toolset-1.52 'bash'
```

2.1. CARGO DIRECTORY STRUCTURE AND FILE PLACEMENTS

The Cargo build tool uses set conventions for defining the directory structure and file placement within a Cargo package. Running the `cargo new` command generates the package directory structure and templates for both a manifest and a project file. By default, it also initializes a new Git repository in the package root directory.

For a binary program, Cargo creates a directory `project_name` containing a text file named `Cargo.toml` and a subdirectory `src` containing a text file named `main.rs`.

Additional resources

- For more information on the Cargo directory structure, see [The Cargo Book — Package Layout](#).
- For in-depth information about Rust code organization, see [The Rust Programming Language — Managing Growing Projects with Packages, Crates, and Modules](#).

2.2. CREATING A RUST PROJECT

Create a new Rust project that is set up according to the Cargo conventions with the `cargo new` command. To edit the project code, edit the main executable file `main.rs` and add new source files to the `src` subdirectory. For more information on Cargo conventions, see Cargo directory structure and file placements.

**Procedure**

- Create a Rust project by running the following command:
  - On Red Hat Enterprise Linux 7:
    ```
    $ scl enable rust-toolset-1.52 'cargo new --bin <project_name>'
    ```
    - Replace `<project_name>` with your project name.
  - On Red Hat Enterprise Linux 8:
    ```
    $ cargo new --bin <project_name>
    ```
    - Replace `<project_name>` with your project name.
2.3. CREATING A LIBRARY FOR A RUST PROJECT

Complete the following steps to create a library for your Rust project using the Cargo build tool.

Prerequisites

- An existing Rust project.
  For information on how to create a Rust project, see Creating a Rust project.

Procedure

- To create a library for your Rust project, run the following command:
  - On Red Hat Enterprise Linux 7:
    ```
    $ scl enable rust-toolset-1.52 'cargo new --lib <project_name>'
    ```
    - Replace `<project_name>` with the name of your Rust project.
  - On Red Hat Enterprise Linux 8:
    ```
    $ cargo new --lib <project_name>
    ```
    - Replace `<project_name>` with the name of your Rust project.

2.4. BUILDING A RUST PROJECT

Build your Rust project using the Cargo build tool. Cargo resolves all dependencies of your project, downloads missing dependencies, and compiles it using the `rustc` compiler.

By default, projects are built and compiled in debug mode. For information on compiling your project in release mode, see Building a Rust project in release mode.

Prerequisites

- An existing Rust project.
  For information on how to create a Rust project, see Creating a Rust project.

Procedure

- To build a Rust project managed by Cargo, run in the project directory:
  - On Red Hat Enterprise Linux 7:
    ```
    $ scl enable rust-toolset-1.52 'cargo build'
    ```
  - On Red Hat Enterprise Linux 8:
    ```
    $ cargo build
    ```
Verification

- To verify that a Rust program managed by Cargo can be built, run in the project directory:
  - On Red Hat Enterprise Linux 7:
    ```sh
    $ scl enable rust-toolset-1.52 'cargo check'
    ```
  - On Red Hat Enterprise Linux 8:
    ```sh
    $ cargo check
    ```

TIP

Consider using the `cargo check` command instead of the `cargo build` command to verify that your Rust program can be built when you do not need to build an executable file. The `cargo check` command is faster than a full project build using the `cargo build` command.

### 2.5. BUILDING A RUST PROJECT IN RELEASE MODE

Build your Rust project in release mode using the Cargo build tool. Release mode is optimizing your source code and can therefore increase compilation time while ensuring that the compiled binary will run faster. Use this mode to produce optimized artifacts suitable for release and production. Cargo resolves all dependencies of your project, downloads missing dependencies, and compiles it using the `rustc` compiler.

For information on compiling your project in debug mode, see [Building a Rust project](#).

**Prerequisites**

- An existing Rust project.
  For information on how to create a Rust project, see [Creating a Rust project](#).

**Procedure**

- To build the project in release mode, run:
  - On Red Hat Enterprise Linux 7:
    ```sh
    $ scl enable rust-toolset-1.52 'cargo build --release'
    ```
  - On Red Hat Enterprise Linux 8:
    ```sh
    $ cargo build --release
    ```

**Verification**

- To verify that a Rust program managed by Cargo can be built, run in the project directory:
  - On Red Hat Enterprise Linux 7:
    ```sh
    $ scl enable rust-toolset-1.52 'cargo check'
    ```
On Red Hat Enterprise Linux 8:

$ cargo check

**TIP**

Consider using the `cargo check` command instead of the `cargo build` command to verify that your Rust program can be built when you do not need to build an executable file. The `cargo check` command is faster than a full project build using the `cargo build` command.

### 2.6. RUNNING A RUST PROGRAM

Run your Rust project using the Cargo build tool. Cargo first rebuilds your project and then runs the resulting executable file. If used during development, the `cargo run` command correctly resolves the output path independently of the build mode.

**Prerequisites**

- A built Rust project.
  - For information on how to build a Rust project, see [Building a Rust project](#).

**Procedure**

- To run a Rust program managed as a project by Cargo, run in the project directory:
  - On Red Hat Enterprise Linux 7:
    - $ scl enable rust-toolset-1.52 'cargo run'
  - On Red Hat Enterprise Linux 8:
    - $ cargo run

**NOTE**

If your program has not been built yet, Cargo builds your program before running it.

### 2.7. TESTING A RUST PROJECT

Test your Rust program using the Cargo build tool. Cargo first rebuilds your project and then runs the tests found in the project. Note that you can only test functions that are free, monomorphic, and take no arguments. The function return type must be either () or `Result<(), E>` where `E: Error`.

By default, Rust projects are tested in debug mode. For information on testing your project in release mode, see [Testing a Rust project in release mode](#).

**Prerequisites**

- A built Rust project.
  - For information on how to build a Rust project, see [Building a Rust project](#).
A function marked as a test.
To mark a function as a test in your source code, add the test attribute `#[test]` in front of your function.

**Procedure**

- To run tests for a Rust project managed by Cargo, run in the project directory:
  
  - On Red Hat Enterprise Linux 7:
    
    ```sh
    $ scl enable rust-toolset-1.52 'cargo test'
    ```
  
  - On Red Hat Enterprise Linux 8:
    
    ```sh
    $ cargo test
    ```

**Additional resources**

- For more information on performing tests in your Rust project, see *The Rust Reference — Testing attributes*.

### 2.8. TESTING A RUST PROJECT IN RELEASE MODE

Test your Rust program in release mode using the Cargo build tool. Release mode is optimizing your source code and can therefore increase compilation time while ensuring that the compiled binary will run faster. Use this mode to produce optimized artifacts suitable for release and production. Cargo first rebuilds your project and then runs the tests found in the project. Note that you can only test functions that are free, monomorphic, and take no arguments. The function return type must be either `()` or `Result<(), E>` where `E: Error`.

For information on testing your project in debug mode, see *Testing a Rust project*.

**Prerequisites**

- A built Rust project.
  For information on how to build a Rust project, see *Building a Rust project*.

- A function marked as a test.
  To mark a function as a test in your source code, add the test attribute `#[test]` in front of your function.

**Procedure**

- To run tests for a Rust project managed by Cargo in release mode, run in the project directory:
  
  - On Red Hat Enterprise Linux 7:
    
    ```sh
    $ scl enable rust-toolset-1.52 'cargo test --release'
    ```
  
  - On Red Hat Enterprise Linux 8:
    
    ```sh
    $ cargo test --release
    ```
2.9. CONFIGURING RUST PROJECT DEPENDENCIES

Configure the dependencies of your Rust project using the Cargo build tool. To specify dependencies for a project managed by Cargo, edit the file Cargo.toml in the project directory and rebuild your project. Cargo downloads the Rust code packages and their dependencies, stores them locally, builds all of the project source code including the dependency code packages, and runs the resulting executable.

Prerequisites

- A built Rust project.
  For information on how to build a Rust project, see Building a Rust project.

Procedure

1. In your project directory, open the file Cargo.toml.

2. Move to the section labelled [dependencies].
   Each dependency is listed on a new line in the following format:

   ```
crate_name = version
```

   Rust code packages are called crates.

3. Edit your dependencies.

4. Rebuild your project by running:
   - On Red Hat Enterprise Linux 7:
     ```
     $ scl enable rust-toolset-1.52 'cargo build'
     ```
   - On Red Hat Enterprise Linux 8:
     ```
     $ cargo build
     ```

5. Run your project by using the following command:
   - On Red Hat Enterprise Linux 7:
     ```
     $ scl enable rust-toolset-1.52 'cargo run'
     ```
   - On Red Hat Enterprise Linux 8:
     ```
     $ cargo run
     ```

Additional resources

- For more information on configuring Rust dependencies, see The Cargo Book — Specifying Dependencies.
2.10. BUILDING DOCUMENTATION FOR A RUST PROJECT

Use the Cargo tool to generate documentation from comments in your source code that are marked for extraction. Note that documentation comments are extracted only for public functions, variables, and members.

Prerequisites

- A built Rust project. For information on how to build a Rust project, see Building a Rust project.
- Configured dependencies. For more information on configuring dependencies, see Configuring Rust project dependencies.
- Comments in your source code marked for extraction into documentation. To mark comments for extraction, use three slashes /// and place your documentation comment in the beginning of the line it is documenting. Cargo supports the Markdown language for your comments.

Procedure

- To build project documentation using Cargo, run in the project directory:
  - On Red Hat Enterprise Linux 7:
    ```bash
    $ scl enable rust-toolset-1.52 'cargo doc --no-deps'
    ```
  - On Red Hat Enterprise Linux 8:
    ```bash
    $ cargo doc --no-deps
    ```

  NOTE

  To include dependencies in the generated documentation, omit the --no-deps option.
  To open the generated documentation in your browser after creation, add the --open option.

Additional resources

- For more information on building documentation using Cargo, see The Rust Programming Language — Making Useful Documentation Comments.

2.11. VENDORING RUST PROJECT DEPENDENCIES

Create a local copy of the dependencies of your Rust project for offline redistribution and reuse using the Cargo build tool. This procedure is called vending project dependencies. The vendored dependencies including Rust code packages for building your project on a Windows operating system are located in the vendor directory. Vendored dependencies can be used by Cargo without any connection to the internet.

Prerequisites
A built Rust project.
For information on how to build a Rust project, see Building a Rust project.

Configured dependencies.
For more information on configuring dependencies, see Configuring Rust project dependencies.

Procedure

To vendor your Rust project with dependencies using Cargo, run in the project directory:

- On Red Hat Enterprise Linux 7:
  $ scl enable rust-toolset-1.52 'cargo vendor'

- On Red Hat Enterprise Linux 8:
  $ cargo vendor

2.12. ADDITIONAL RESOURCES

- For more information on Cargo, see the Official Cargo Guide.

- To display the manual page included in Rust Toolset, run:
  
  - For Red Hat Enterprise Linux 7:
    $ scl enable rust-toolset-1.52 'man cargo'

  - For Red Hat Enterprise Linux 8:
    $ man cargo
CHAPTER 3. THE RUSTFMT FORMATTING TOOL

With the `rustfmt` formatting tool, you can automatically format the source code of your Rust programs. You can use `rustfmt` either as a standalone tool or with Cargo.

NOTE
To avoid using `scl enable` in front of every command, run:

```bash
$ scl enable rust-toolset-1.52 'bash'
```

3.1. INSTALLING RUSTFMT

Complete the following steps to install the `rustfmt` formatting tool.

Prerequisites
- Rust Toolset is installed. For more information, see Installing Rust Toolset.

Procedure
- Run the following command to install `rustfmt`:
  - **On Red Hat Enterprise Linux 7:**
    ```bash
    # yum install rust-toolset-1.52-rustfmt
    ```
  - **On Red Hat Enterprise Linux 8:**
    ```bash
    # yum install rustfmt
    ```

3.2. USING RUSTFMT AS A STANDALONE TOOL

Use `rustfmt` as a standalone tool to format a Rust source file and all its dependencies. As an alternative, use `rustfmt` with the Cargo build tool. For more information, see Using rustfmt with Cargo.

Prerequisites
- An existing Rust project. For information on how to create a Rust project, see Creating a Rust project.

Procedure
- To format a Rust source file using `rustfmt` as a standalone tool, run the following command:
  - **On Red Hat Enterprise Linux 7:**
    ```bash
    $ scl enable rust-toolset-1.52 'rustfmt <source-file>'
    ```
- Replace `<source_file>` with the name of your source file. Alternatively, you can replace `<source_file>` with standard input. `rustfmt` then provides its output in standard output.

  - On Red Hat Enterprise Linux 8:
    ```bash
    $ rustfmt <source-file>
    ```

- Replace `<source_file>` with the name of your source file. Alternatively, you can replace `<source_file>` with standard input. `rustfmt` then provides its output in standard output.

**NOTE**

By default, `rustfmt` modifies the affected files without displaying details or creating backups. To display details and create backups, run `rustfmt` with the `--write-mode value`.

  - On Red Hat Enterprise Linux 7:
    ```bash
    $ scl enable rust-toolset-1.52 'rustfmt <source_file>' --write-mode value
    ```

  - On Red Hat Enterprise Linux 8:
    ```bash
    $ rustfmt <source_file> --write-mode value
    ```

### 3.3. USING RUSTFMT WITH CARGO

Use the `rustfmt` tool with Cargo to format a Rust source file and all its dependencies. As an alternative, use `rustfmt` as a standalone tool. For more information, see Using `rustfmt` as a standalone tool.

**Prerequisites**

- An existing Rust project.
  For information on how to create a Rust project, see Creating a Rust project.

**Procedure**

- To format all source files in a Cargo code package, run the following command:
  
  - On Red Hat Enterprise Linux 7:
    ```bash
    $ scl enable rust-toolset-1.52 'cargo fmt'
    ```

  - On Red Hat Enterprise Linux 8:
    ```bash
    $ cargo fmt
    ```

**NOTE**

To change the `rustfmt` formatting options, create the configuration file `rustfmt.toml` in the project directory and add your configurations to the file.
3.4. ADDITIONAL RESOURCES

- To display the help pages of `rustfmt`, run:
  - On Red Hat Enterprise Linux 7:
    
    ```bash
    $ scl enable rust-toolset-1.52 'rustfmt --help'
    ```
  - On Red Hat Enterprise Linux 8:
    
    ```bash
    $ rustfmt --help
    ```

- To configure the `rustfmt` tool, edit the file `Configurations.md`.
  - On Red Hat Enterprise Linux 7, find it under the following path:
    `/opt/rh/rust-toolset-1.52/root/usr/share/doc/rust-toolset-1.52-rustfmt-1.52.1/Configurations.md`
  - On Red Hat Enterprise Linux 8, find it under the following path:
    `/usr/share/doc/rustfmt/Configurations.md`
CHAPTER 4. CONTAINER IMAGES WITH RUST TOOLSET

You can build your own Rust Toolset container images on top of Red Hat Universal Base Images (UBI) containers using Dockerfiles.

4.1. CREATING A CONTAINER IMAGE OF RUST TOOLSET

Rust Toolset packages are part of the Red Hat Universal Base Images (UBIs) repositories. To keep the container size small, install only individual packages instead of the entire Rust Toolset.

Prerequisites

- An existing Dockerfile. For more information on creating Dockerfiles, see the Dockerfile reference page.
- A set up UBI.
  a. To set up a UBI, visit the Red Hat Container Catalog.
    - On Red Hat Enterprise Linux 8, search for UBI 8.
  b. Select a UBI.
  c. Click Get this image and follow the instructions.

Procedure

- To create a container containing Rust Toolset, add the following lines to your Dockerfile:
  - On Red Hat Enterprise Linux 7:
    ```
    FROM registry.access.redhat.com/ubi7/ubi:latest
    RUN yum install -y rust-toolset-1.52
    ```
  - On Red Hat Enterprise Linux 8:
    ```
    FROM registry.access.redhat.com/ubi8/ubi:latest
    RUN yum install -y rust-toolset
    ```
- To create a container image containing an individual package only, add the following lines to your Dockerfile:
  - On Red Hat Enterprise Linux 7:
    ```
    RUN yum install rust-toolset-1.52-<package-name>
    ```
    - Replace `<package-name>` with the name of the package you want to install.
  - On Red Hat Enterprise Linux 8:
    ```
    RUN yum install <package-name>
    ```
Replace `<package_name>` with the name of the package you want to install.

### 4.2. ADDITIONAL RESOURCES

- For more information on Red Hat UBI images, see *Working with Container Images*.
- For more information on Red Hat UBI repositories, see *Universal Base Images (UBI): Images, repositories, packages, and source code*. 
Rust Toolset rebased to version 1.52.1

Rust Toolset has been updated from version 1.49.0 to 1.52.1. Notable changes include:

- It is now possible to use constant-value parameters to define generics. With this change, you can write functions completely generic over the values of any integer, boolean, or character type, and arrays generic over their element type as well as their length. Moreover, it is now possible to iterate items from an array by value using the new standard library’s array type API std::array::IntoIter.

- With this release, incremental compilation is disabled by default. A new verification added in Rust 1.52.0 detected a bug present in all rust versions that can trigger miscompilations in incremental builds. Since downgrading does not fix this insecurity, either disable incremental compilation or upgrade to Rust 1.52.1.

For detailed information regarding the updates, see the upstream Rust 1.52.1 Release Notes.