Live Workshop: Building Your First Stylus Project

1. Creating Your First Stylus Project

Begin by creating your first Stylus project:

```
cargo stylus new my_stylus_counter
```

This generates a Rust starter template similar to the Solidity Counter example. The goal is a simple contract:

- `setCount`: increments the counter
- `getCount`: retrieves the current count

lib.rs (Generated Template)

Here's the generated Rust equivalent of the Solidity Counter example:

Line-by-Line Explanation:

- `#![cfg_attr(not(any(test, feature = "export-abi")), no_main)]`: Specifies the absence of the main function when not testing or exporting ABI.
- `#![cfg_attr(not(any(test, feature = "export-abi")), no_std)]`: Declares that the standard library is not used, common in embedded or blockchain environments.
- `extern crate alloc;`: Imports Rust's allocator, used when `no_std` is active.
- `use alloc::vec::Vec;`: Imports Vec from the alloc crate for dynamic arrays.
- `use stylus_sdk::{alloy_primitives::U256, prelude::*};`: Imports essential Stylus SDK components including the U256 type for large integer operations.

Storage definition:

```
sol_storage! {
    #[entrypoint]
    pub struct Counter {
        uint256 number;
    }
}
```

- Defines a storage struct with a single `uint256` value named `number`.

Methods implementation:

```
#[public]
impl Counter {
```

- `#[public]`: Makes functions publicly accessible.

Individual methods:

- `number()`: Getter method retrieving the current number stored.
- `set_number(new_number: U256)`: Setter method updating the stored number.
- `mul_number(new_number: U256)`: Multiplies current stored number by provided value.
- `add_number(new_number: U256)`: Adds provided value to current stored number.
- `increment()`: Increments the stored number by one.
- `add_from_msg_value()`: Adds transaction's `msg.value` to the stored number (Ethereum-specific concept).

Test module:

```
#[cfg(test)]
mod test {
```

- Tests validate each method ensuring correct behavior.

Ethereum & Solana Reference:

- **Ethereum Devs:** Familiar EVM patterns—state management, getters/setters, and payable functions.
- **Solana Devs:** Reflects Solana's smart contract (program) account-data updates, with Rust syntax and structure very close to Anchor-based smart contracts.

Audience Questions & Answers:

- **Q:** Why does Stylus use Rust instead of Solidity directly?

- **A:** Rust provides enhanced performance, security, and memory management advantages suitable for complex computations compared to Solidity.

- **Q:** How do I debug my Stylus contract if `cargo stylus check` fails?

- **A:** Check the detailed error output, review reserved keywords, ensure Docker is running, and revisit the Stylus SDK documentation.

- **Q:** Can I integrate Stylus contracts with existing Ethereum dApps?

- **A:** Yes, Stylus contracts are EVM-compatible, allowing seamless integration with Ethereum tools and dApps.

- **Q:** Does deploying Stylus contracts cost more or less gas compared to regular Solidity contracts?

- **A:** Generally less. WASM contracts typically execute more efficiently, using fewer resources and thus lower gas.

---

2. Validating Your Stylus Project

Run a dry-check to ensure the contract compiles and is valid for deployment:

```
cargo stylus check
```

Important: Ensure Docker is running.

- A successful check indicates your contract is deployable.
- Errors provide insights into fixes needed, such as reserved symbol usage or WASM parsing issues.

---

3. Deploying Your Stylus Contract

Estimate gas required before deploying:

```
cargo stylus deploy \
  --endpoint='http://localhost:8547' \
  --private-key="YOUR_PRIVATE_KEY" \
  --estimate-gas
```

Then, deploy the contract:

```
cargo stylus deploy \
  --endpoint='http://localhost:8547' \
  --private-key="YOUR_PRIVATE_KEY"
```

- Save the contract deployment address provided.

---

4. Exporting Solidity ABI

Generate the ABI for dApp integration:

```
cargo stylus export-abi
```

Save this ABI output to a file for future interactions with Ethereum-compatible tools.

---

5. Interacting With Your Deployed Contract

Use Foundry's Cast to interact:

- **Reading the Counter Value:**

```
cast call --rpc-url 'http://localhost:8547' \
--private-key YOUR_PRIVATE_KEY \
<deployed-contract-address> "number()(uint256)"
```

Initially, this returns `0`.

- **Incrementing the Counter:**

```
cast send --rpc-url 'http://localhost:8547' \
--private-key YOUR_PRIVATE_KEY \
<deployed-contract-address> "increment()"
```

Verify increment by calling `number()(uint256)` again.

---

6. Conclusion

Congratulations! You've successfully:

- Initialized and validated your Stylus project
- Deployed your contract to Arbitrum
- Interacted using Ethereum-compatible tools

You're now equipped to build more advanced Stylus contracts!