Entries

main
aptos-node/src/main.rs

AptosNodeArgs::parse().run()

run
aptos-node/src/lib.rs

start(config, None, true).expect("Node should start correctly");

start_and_report_ports
aptos-node/src/lib.rs

let _node_handle = setup_environment_and_start_node(

setup_environment_and_start_node
aptos-node/src/lib.rs

let mut admin_service = services::start_admin_service(&node_config);
storage::initialize_database_and_checkpoints(&mut node_config)?;
admin_service.set_aptos_db(db_rw.clone().into());
utils::set_aptos_vm_configurations(&node_config);
let telemetry_runtime = services::start_telemetry_service(
create_event_subscription_service(&node_config, &db_rw);
network::create_peers_and_metadata(&node_config);
network::setup_networks_and_get_interfaces(
services::start_peer_monitoring_service(
state_sync::start_state_sync_and_get_notification_handles(
services::start_node_inspection_service(
services::bootstrap_api_and_indexer(
admin_service.set_mempool_client_sender(mempool_client_sender);
services::start_mempool_runtime_and_get_consensus_sender
consensus::create_dkg_runtime
consensus::create_jwk_consensus_runtime
state_sync_runtimes.block_until_initialized();
consensus::create_consensus_observer_and_publisher(
consensus::create_consensus_runtime(

state_sync

create_event_subscription_service
aptos-node/src/state_sync.rs

EventSubscriptionService::new(Arc::new(RwLock::new(db_rw.clone())));
let mempool_reconfig_subscription = event_subscription_service
.subscribe_to_reconfigurations()
let consensus_observer_reconfig_subscription =event_subscription_service
.subscribe_to_reconfigurations()
let consensus_reconfig_subscription = event_subscription_service
.subscribe_to_reconfigurations()
let dkg_subscriptions =
    let reconfig_events = event_subscription_service
    .subscribe_to_reconfigurations()
    let dkg_start_events = event_subscription_service
    .subscribe_to_events
let jwk_consensus_subscriptions =
    let reconfig_events = event_subscription_service
    .subscribe_to_reconfigurations()
    let dkg_start_events = event_subscription_service
    .subscribe_to_events

start_state_sync_and_get_notification_handles
aptos-node/src/state_sync.rs

setup_aptos_data_client(node_config, network_client,
db_rw.reader.clone())?;
setup_data_streaming_service(state_sync_config,
aptos_data_client.clone())?;
aptos_mempool_notifications::new_mempool_notifier_listener_pair(
aptos_consensus_notifications::new_consensus_notifier_listener_pair(
aptos_storage_service_notifications::new_storage_service_notifier_list
ener_pair();
let storage_service_runtime = setup_state_sync_storage_service(
let state_sync = DriverFactory::create_and_spawn_driver(

event-notifications

subscribe_to_reconfigurations
state-sync/inter-component/event-notifications/src/lib.rs

if self.reconfig_subscriptions
    .insert(subscription_id, reconfig_subscription)

subscribe_to_events
state-sync/inter-component/event-notifications/src/lib.rs

if let Some(old_subscription) = self
    .subscription_id_to_event_subscription
    .insert(subscription_id, event_subscription)
for event_key in event_keys {
    self.event_key_subscriptions
        subscriptions.insert(subscription_id);
for event_tag in event_v2_tags {
    self.event_v2_tag_subscriptions
        subscriptions.insert(subscription_id);

create_and_spawn_driver
state-sync/state-sync-driver/src/driver_factory.rs

let (driver_factory, _) = Self::create_and_spawn_driver_internal(

create_and_spawn_driver_internal
state-sync/state-sync-driver/src/driver_factory.rs

let state_sync_driver = StateSyncDriver::new(
tokio::spawn(state_sync_driver.start_driver());

state_sync_driver

start_driver
state-sync/state-sync-driver/src/driver.rs

loop {
    notification = self.client_notification_listener.select_next_some() => {
        self.handle_client_notification(notification).await;
    notification = self.commit_notification_listener.select_next_some() => {
        self.handle_snapshot_commit_notification(notification).await;
    notification = self.consensus_notification_handler.select_next_some() => {
        self.handle_consensus_or_observer_notification(notification).await;
    notification = self.error_notification_listener.select_next_some() => {
        self.handle_error_notification(notification).await;

handle_consensus_or_observer_notification
state-sync/state-sync-driver/src/driver.rs

match notification {
    ConsensusNotification::NotifyCommit(commit_notification) => {
        .respond_to_commit_notification
    ConsensusNotification::SyncToTarget(sync_notification) => {
        .respond_to_sync_target_notification
    ConsensusNotification::SyncForDuration(sync_notification) => {
        .respond_to_sync_duration_notification

let result = match notification {
    ConsensusNotification::NotifyCommit(commit_notification) => {
        self.handle_consensus_commit_notification(commit_notification)
    ConsensusNotification::SyncToTarget(sync_notification) => {
        self.handle_consensus_sync_target_notification(sync_notification)
    ConsensusNotification::SyncForDuration(sync_notification) => {
        self.handle_consensus_sync_duration_notification(sync_notification)

handle_consensus_commit_notification
state-sync/state-sync-driver/src/driver.rs

utils::handle_committed_transactions(
    self.mempool_notification_handler.clone(),
    self.event_subscription_service.clone(),
    self.consensus_notification_handler
        .respond_to_commit_notification(commit_notification, Ok(()))?;

handle_committed_transactions
state-sync/state-sync-driver/src/driver.rs

utils::handle_committed_transactions(

handle_transaction_notification
state-sync/state-sync-driver/src/notification_handlers.rs

event_subscription_service
    .notify_events(latest_synced_version, events)?;

progress_check_interval in driver to handle the interval of each event dispatch